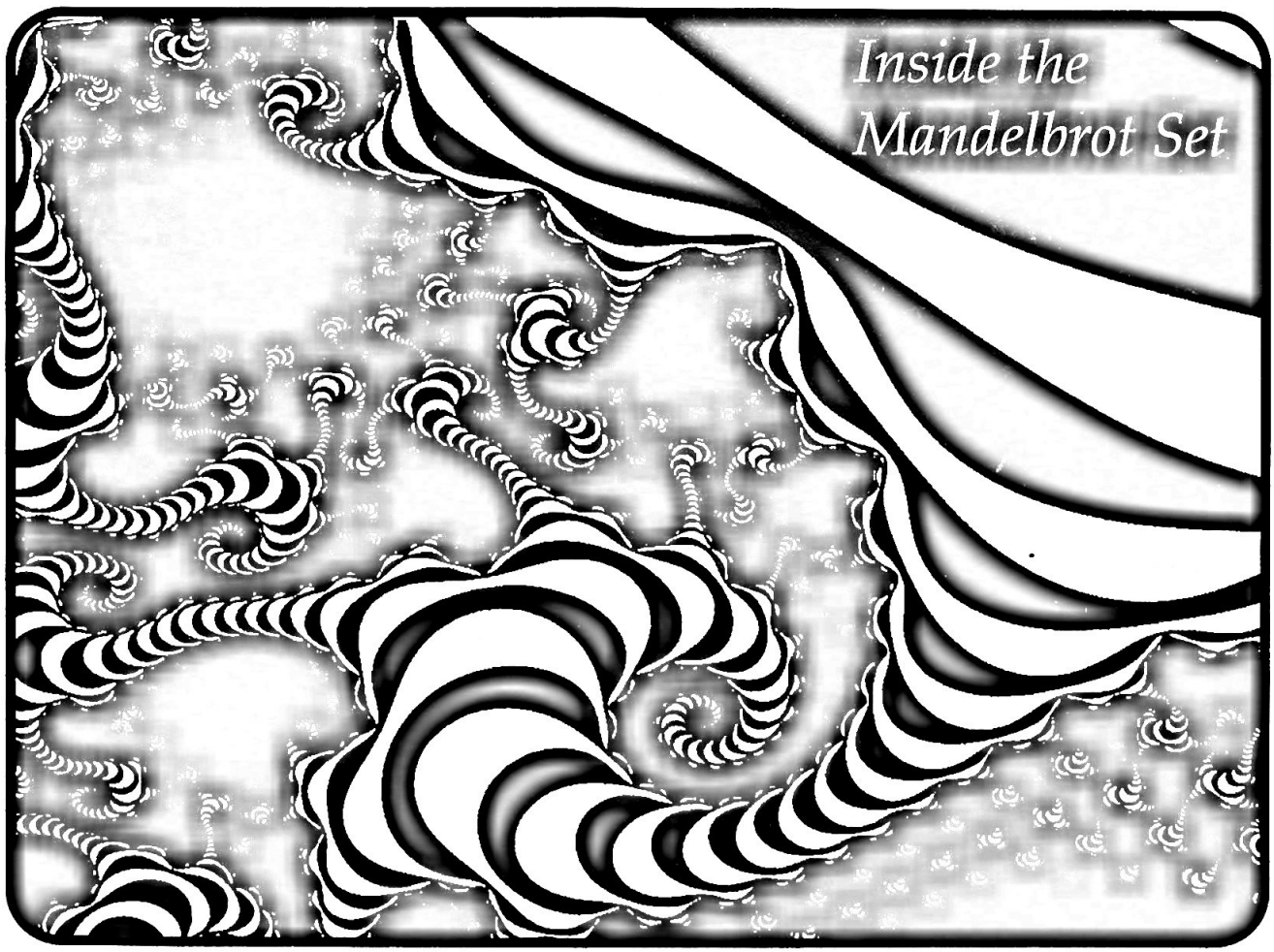
NOVEMBER / DECEMBER 1989

Algorithm 1.1

THE PERSONAL PROGRAMMING NEWSLETTER

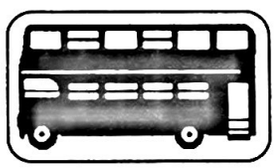


The Emergence of Artificial Life
Hailstone Numbers: Ups and Downs
The Program That Ran a Business

WHO READS Algorithm ?

systems managers, doctors, computer hobbyists, application programmers, high school students, lawyers, professors, hackers, airline pilots, engineers, researchers, teachers, stock brokers, university students, philosophers, reporters, entrepreneurs, able seamen, dreamers, technicians, game-designers, accountants, social workers, movers, shakers and candlestick makers.

WHO READS **Advertising**?



TURING OMNIBUS PRESENTS

EXPLOR-I

CUSTOM PACKAGES FOR EXPERIMENT & ADVENTURE!

Mandelbrot & Julia Sets, Chaotic Systems, Mental Wallpaper, Star Clusters, Biomorphs, Mathematical curves, Cyclic Space, Life and other education programs.

Write for our free catalogue: Box 1456, London, Ontario, Canada, N6A 5MZ

Algorithm

THE PERSONAL PROGRAMMING NEWSLETTER

ALGOLETTER.....2

A.K. Dewdney

An introduction to the newsletter that celebrates the joys of programming.

EASY PIECES.....5

Michael W. Ecker

Beginners compute hailstone numbers in the course of learning some basic BASIC.

PERSONAL PROGRAMS.....9

Clifford A. Pickover

Structures inside the Mandelbrot set rival those on its boundary.

ALGOFACT: Artificial Life.....13

Christopher G. Langton

The search for artificial systems with emergent, lifelike properties.

ALGOFICTION: Alphie and Omega.....19

Anthony Zweistein

The imitation game takes a sinister turn: Was Mr. Omega a person or a program?

BULLETIN BOARD.....24

Software , games and publications available from individuals and small companies.

Algostaff

Editor
A.K. Dewdney

Managing Editor
Louis D. Magguilli

Assistant Editor
Jennifer Connor

Business Manager
Nancy Z. Magguilli

Layout & Design
Tammy Golinski

Production Services
Two Oxford West

Address all editorial correspondence to A.K. Dewdney, 2 Oxford Street West, London, Ontario N6H 1P9, CANADA. Address advertising, business and all other correspondence to Louis D. Magguilli, Algorithm, P.O. Box 29237, Westmount Postal Outlet, 785 Wonderland Road S., London, Ontario N6K 1M6. Owned and published bimonthly by 854832 Ontario Inc., doing business as *Algorithm*.

All rights reserved, including the right to accept or reject all editorial and advertising matter. Unsolicited materials must be accompanied by self-addressed return envelopes and postage or dollar equivalent. No part of this newsletter may be reproduced without the written permission of *Algorithm*. Copyright© 1989 by *Algorithm*.

Mail subscriptions: $29.95 (U.S.) per year single copies $5.00 (U.S.).

ALGOLETTER

by A.K. Dewdney

Introducing Algorithm, The Personal Programming Newsletter

In subsequent issues of our fledgling publication, the Algoletter will be written by readers with something interesting to say on the subject of computing. But the Algoletter of the inaugural issue seems the perfect platform for the editor to introduce his themes.

Our Personal Programming Newsletter has its roots in the Computer Recreations column of Scientific American magazine. I hereby invite both my regular readers and others to join with us in the enlarged personal programming adventure.

How does personal programming differ from personal computing? Originally, the two terms were practically synonymous. Computing is never more personal than when we are writing programs. In recent years the enterprise has slipped sideways into a rut of passivity. Personal computing has been compromised and, finally, dominated by software that is sold from the pages of every computer magazine on the newstand. Readers are rarely urged to write their own programs or even to think over substantive issues and ideas connected with the enterprise. Much less do the few exercises that remain pursue knowledge and pleasure mingled in recreational disguise.

The word "Algorithm" in our five-word title signals the return of the active component to personal computing. An algorithm is not, after all, a program but the outline of one. It focuses thought on the essential process while leaving the implementation to the reader. As far as readers are concerned, both professional programmers and beginners may expect to find excitement in our pages. Even for non-programmers there are articles that demand no more than a modicum of computer literacy.

We shall eschew, in our columns, articles and advertising, the passive aspect of personal computing, namely the use of programs written by others.

Such a policy obviously has its limits. Everyone with a serious interest in computing inevitably uses programs written by others, not merely utilities, word-processors, and spreadsheets, but games, educational programs, simulations, scientific and business applications, and so on. The point is not to avoid the programs of others but to enjoy the ultimate power of a computer system. But, consistent with our policy, Algorithm will avoid lengthy product reviews, encourage the small producers of software having an educational and/or recreational bent, and advertise those products from the larger world that have a direct bearing on the personal programming enterprise: hardware, language systems, books, and so on.

The positive side of personal programming as presented in these pages will hinge on five features which I will now introduce.

Following this Algoletter, neophyte programmers will find a column called Easy Pieces. Written by Dr. Michael W. Ecker, a professor of Computer Science at Pennsylvania State University, Easy Pieces uses intriguing topics from the world of science and mathematics to spark the construction of a simple program. Sometimes, as in this issue, it will spell out a program in BASIC. In other issues it may give only a part of such a program, with the rest left tantalizingly at the algorithmic level. We view this column as the doorway to that closet where some people still keep their computers. Dr. Ecker has written about computing for many years. He has been a columnist in the famous old Creative Computing magazine and continues with a newsletter of his own called REC: Recreational and Educational Computing.

For readers with somewhat more developed skills there is the Personal Programs feature, a column with much the same modus operandi as Easy Pieces but pitched a little higher. How much higher? That will vary: sometimes barely advanced beyond an Easy Piece, sometimes well beyond that level. Dr. Clifford Pickover is known to many as a bold and innovative explorer of graphic forms. He works as a researcher at the Thomas J. Watson Research Center in Yorktown Heights, N.Y. and he edits the Journal of Chaos and Graphics.

In both columns, our core offerings, the emphasis is on the substance behind a program as well as the program itself. We subscribe to the notion that a program should do something interesting or useful. There is a world of ideas—scientific, mathematical, sociological, recreational, even artistic—that we are free to address in parallel to the programming enterprise. A game of hangman is not what we have in mind.

Algofact is a one-article department that is visited each issue by some well-known computer professional. Take warning at the word "professional": it could mean a scientist (as it does this issue), a writer of computer games, an engineer--in short, any outstanding representative of the computational side of virtually any field that has a computational side. The Algofact article in this issue comes from Dr. Christopher G. Langton, a researcher in the field of Artificial Life at the Los Alamos National Laboratory in Los Alamos, New Mexico. Langton has been the moving spirit behind the creation of a new area of inquiry, Artificial Life. Are the goals of this project reasonable? Langton spells them out in a special article on the new discipline.

There is always some danger, especially in areas where speculation is rife, that thinkers get a little beyond their evidence and begin to make claims that try our credulity. Everyone can think of publications, even ones allegedly devoted to science, where no distinction is made between what is solid and what is (at best) speculative. Algorithm has taken a pledge not to so mislead its readers.

As if to underline the commitment, we have included the Algofiction section. Fact and fiction, as nearly as we can distinguish them, are thereby set apart.

At the end of the Newsletter, readers will find a Bulletin Board that has retreated from the electronic form back to print. It brings the work of individuals and small companies to our notice. All of them have interesting things to distribute, things that many readers of Computer Recreations have asked me for in the past: a Mandelbrot Set program, an intelligent game, a learning experience, a book, a newsletter. Much of the material appearing here comprises a kind of "underground economy" never before accessible in one place. Readers who have done something to be proud of may advertise their wares here with some hope of contacting kindred spirits. What other readers buy from the Bulletin Board they may not have the time or energy to produce for themselves.

Both in the features included here and those yet to come, Algorithm offers a unique window on the world of computing. The action is where the action has been all along. In the mind.

A.K. Dewdney
Editor

The Bulletin Board

The Bulletin Board is an issue-by-issue directory of products and services provided for our readers by indivuals and small companies or organizations. We will accept ads for software of educational or recreational value, pamphlets, books, videotapes, clubs, organizations and so on.

To place a Bulletin Board item, send camera-ready copy in the proportions of the rectangle on the back page of the newsletter–or fill this space out by hand and we will set it in type. Be sure to include all relevant information such as address and conditions of distribution.

First placements are free but subsequent placements of an ad (or its equivalent) will be charged at the rate of $30.00 (U.S.) per issue.

ERRATA

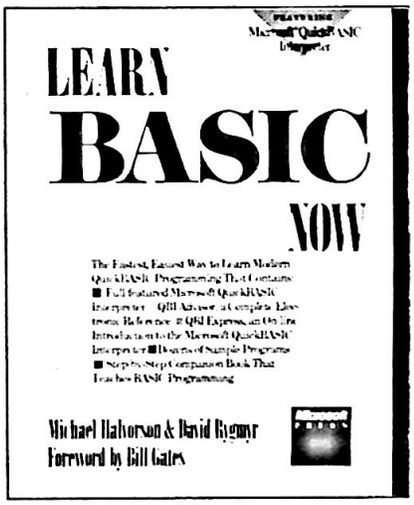
Mandelbrot Algorithm

```
for cx <— -2 to + 2 by steps of .01;
   for cy <— -2 to + 2 by steps of .01;
      x <— 0; y <— 0;
      for i <— 1 to 50;
       xx <— x² - y² + cx;
       y <— 2xy + cy;
       x <— xx;
        if x² + y² > 4  then exit loop
      endloop
      color <— i
      plot point (cx,cy) wth color
   endloop
endloop
```

Some last minute changes at press time went uncorrected. We apologize in the expectation that future issues will be relatively bug-free as we iron the wrinkles out of production. The most crucial corrections are to the box containing the Mandelbrot algorithm on page 11. The corrected algorithm is shown to the left. Our ISSN #0843-9753 was missing from the bottom of page 1. The Algopuzzle solution box on page 18 also contains a number of typographical errors: the puzzle is derived from an interesting and more general problem. The Stevens Institute is in Hoboken, of course.

Learning How to Program Has Never Been Easier!

Now's the time to try your hand at programming. These software-based tutorials—with an excellent combination of working software, electronic help, and book tutorials—make the learning easy. And they're from Microsoft—the software company that leads the field in innovative programming languages. So, now's your chance. Choose a language and start programming!



LEARN BASIC NOW

Michael Halvorson and David Rygmyr; Foreword by Bill Gates

An exciting new approach to learning BASIC programming. LEARN BASIC NOW comes complete with the speedy Microsoft® QuickBASIC Interpreter (a $39.95 value), top-notch electronic help resources, dozens of sample programs, and outstanding on-line and book tutorials. Learn BASIC the QuickBASIC way and discover the advantages of modern programming techniques. Unlike other BASICs, Microsoft QuickBASIC combines powerful, state-of-the-art features—like automatic error checking and correction—with innovations like drop-down menus and on-line help that make BASIC programming easy to learn. If you have little or no programming experience, LEARN BASIC NOW is the fastest, easiest route to understanding programming concepts and writing BASIC programs. For IBM® Personal Computers and Compatibles.

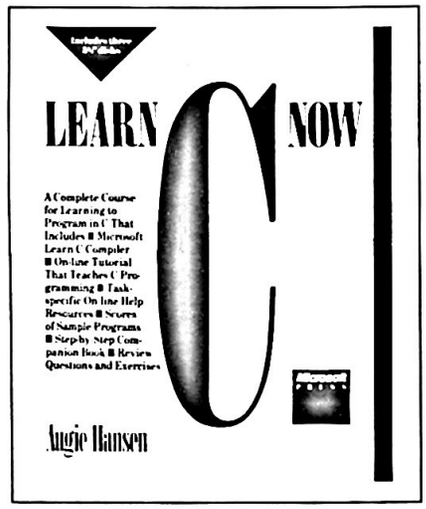
350 pages, 7 3/8 x 9 1/4, softcover with three 5.25-inch disks $39.95

LEARN C NOW

Augie Hansen

Now you can learn C programming even if you have little or no programming experience! LEARN C NOW is a completely integrated self-study course designed to make C programming fun and easy to learn. Everything you need is contained on the three 5 1/4" disks and in the companion book including: the Learn C in-memory compiler (based on Microsoft® QuickC™) complete with full-screen editor, debugger, and blazing 7000-line-per-minute compiler speed ■ the Learn C on-line tutorial with detailed information on using the Learn C compiler and lessons in C programming ■ scores of sample programs ■ the companion book that works hand-in-hand with the on-line tutorial. Seize the opportunity and LEARN C NOW!

384 pages, 7 3/8 x 9 1/4, softcover with three 5.25-inch disks $39.95



Microsoft Press books are available wherever books and software are sold. Or you can order directly from Microsoft Press.

ORDER TODAY

YES! Please send me the following:

_____ copies of LEARN BASIC NOW at $39.95* each
[Order Code 1-55615-240-X] ..... $_____

_____ copies of LEARN C NOW at $39.95* each
[Order Code 1-55615-130-6] ..... $_____

Sales Tax: CA—6.5%; CT—8%; FL—6%; IL—5%; KY—5%; MA—5%; MD—5%; MN—6%; MO—4.425%; NJ—6%; NY—4%; SC—5%; TX—6%; WA state—8.1% ..... $_____

Postage & Handling: $2.50; $.75 for each additional book. ..... $_____

Total ..... $_____

All orders shipped via UPS. Please allow 4 weeks for delivery. For faster service—call in your credit card order. **800-888-3303** (8AM to 5PM PST). Refer to ad: ALGO.
*Prices higher in Canada. For details on ordering outside the U.S. call **800-888-3303.**

NAME _____

ADDRESS _____

_____

CITY _____ STATE _____ ZIP _____

Payment: ☐ Check ☐ VISA ☐ MasterCard ☐ American Express

CREDIT CARD NO. _____ EXP. DATE _____

Send your order and payment or credit card information to:
Microsoft Press
Attn: Consumer Sales Dept.
16011 NE 36th Way, Box 97017
Redmond, WA 98073-9717.

Microsoft® Press
Computer Books That Make Sense

ALGO

EASY PIECES

by Michael W. Ecker

A Maiden Voyage into the Clouds to Look at Hailstone Numbers.

If this first column were a spaceship, we might require a systems check: computer on? check... BASIC loaded?. . . check. The countdown might sound strange, however: 8,4,2,1. This is the sound of a hailstone number coming to earth.

The aim of this flight is to write a program that generates the mysterious hailstone numbers. Easy to generate, they nevertheless have mathematicians scratching their heads. Ask one about the Collatz conjecture, the Syracuse problem or the ulam conjecture, all names for the conundrum that hailstone numbers present us with.

Suppose that we start with any whole number. If the number is even, divide it by 2. But if the number is odd, multiply by 3 and add 1. What happens when you do this repeatedly?

Take the number 5 as an example. Since 5 is odd, the second formula is applied, leading to:

$$3 \times 5 + 1 = 16$$

To 16 we apply the first formula (dividing by 2):

$$16 / 2 = 8$$

Again, 8 leads to 4, which leads to 2 and then to 1. Is this the end? Almost. Since 1 is odd we apply the second formula again to get 4. But 4, as we already know, leads to 1. Here is a cycle 4-2-1 that, once reached, never ends.

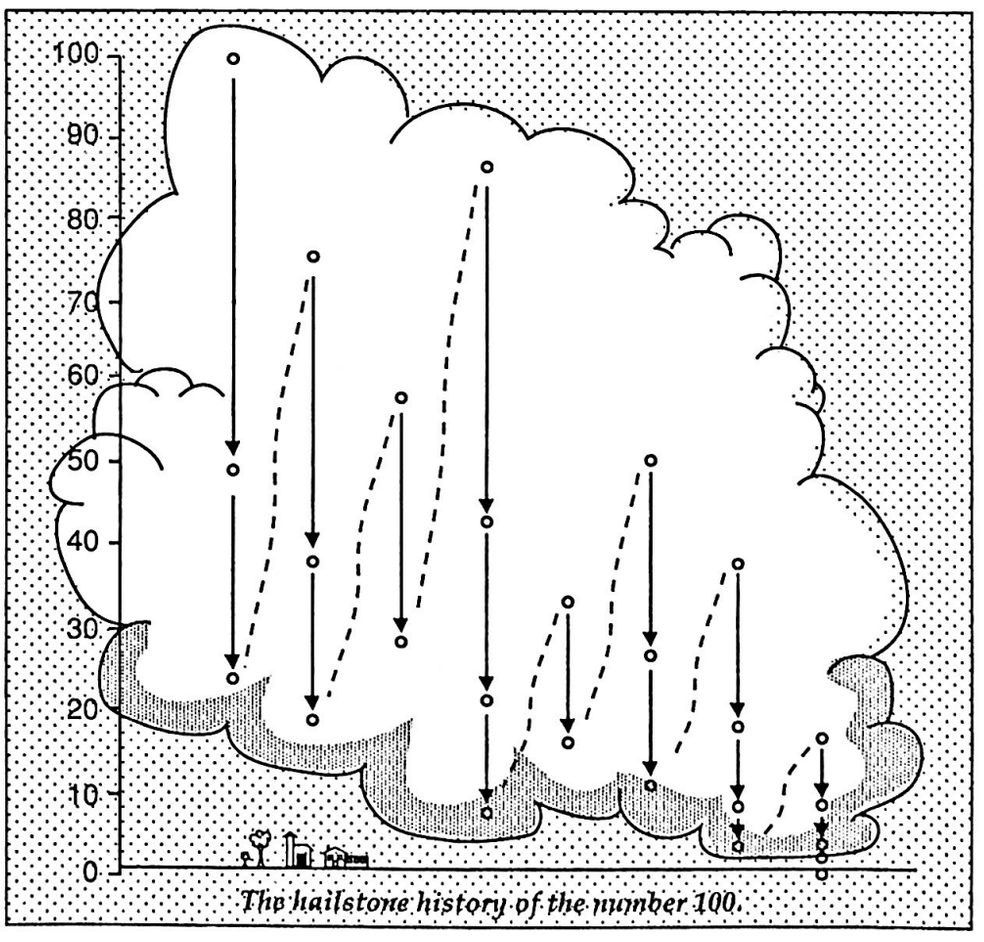
One example proves nothing, of course. What happens if we start with 100 instead? The figure on this page shows the resulting number sequence as though it were the history of a single hailstone, 100, in a cloud full of numerical updrafts. At first 100 drops through 50 down to 25. Then it climbs abruptly to 76, drops through 38 to 19 and thereafter experiences no fewer than 6 ups and 14 downs before "coming to earth" in the cyclic sequence 4-2-1. In all, 25 repetitions or iterations of the basic formulas were required before the hailstone finally came down.

Let's try another example using a larger number, say 123456789. Once you have written a program that, like mine, generates the hailstone sequences automatically, you'll find the same kind of erratic pattern of ups and downs. In this case there are 58 ups and 119 downs for a total of 177 iterations before reaching 1.

The natural question is, "Does every possible initial choice of number lead to the 4-2-1 cycle"? The fact that it did in two cases is encouraging but hardly proof. Although no one has ever found a number that does not eventually lead to this cycle, no one has proven that this must always happen.

Usually, mathematicians favor analysis as the tool of choice in such questions. In spite of the fact that every number that anyone has ever tried leads to the 4-2-1 cycle, no one, mathematician or not, has ever succeeded in proving that this must always happen. Hence the words, "conjecture" and "problem" appearing in connection with this process. Since the 1930s mathematicians have conjectured that the 4-2-1 sequence always results, but proving it turns out to be quite a problem indeed! With the advent of computers, many people have looked for patterns in the ups and downs to gain insight missed by traditional methods.

One traditional insight notes that every odd number in a hailstone sequence is succeeded by an even one. If we knew that this new number were a multiple not just of 2, but 4, we could argue that "3 times ahead and 4 times back" guarantees that the numbers get smaller. By the argument known



The hailstone history of the number 100.

as infinite descent, we could then claim that the numbers would be eventually pulled down into some kind of cycle.

Unfortunately, things are more complicated than that. We can certainly say, "3 times ahead and 2 times back," but what could we conclude? Our hailstone might draft upward, ever diverging to infinity. On the other hand, one of the numbers in the sequence might well be a power of 2 (a number of the form 2 x 2 x . . . x 2). It would then be divided repeatedly by 2 until it arrived at ground level, 4-2-1.

A Hailstone Program

To get a handle on hailstone numbers, it's time to look at some easy programming techniques in Microsoft BASIC. The power of this programming language will serve well, not only with the easy pieces of this column but with more complicated programming problems, whether recreational or serious.

task. Unlike a program, an algorithm's statements do not have to follow a rigid form or syntax; they can be quite informal, as long as their meaning is unambiguous. For example, we could write out informal but perfectly precise directions for making tea that almost anyone could follow.

The algorithm for generating hailstone numbers could be written as follows:

1. Start with a number.
2. Calculate the next number:
   - if the number is even, divide by 2
   - if the number is odd, multiply by 3 and add 1
   - print the number so computed
3. Iterate, that is, repeat step 2 with the new number.

Most experienced programmers draw up some kind of diagram, plan or algorithm before writing their program. A well-written BASIC program, for example, usually results from an algorithmic beginning.

```
10 PRINT  "What is the starting number?"
20 INPUT N
30 IF N/2 = INT(N/2) THEN N = N/2 ELSE N = 3*N + 1
40 PRINT N
50 GOTO 30
60 END
```

Simple Hailstone Program

Shown in the box above is an easy program that will serve our immediate purpose.

One way to understand what the program does is to examine its operation in outline, namely, as an algorithm.

What is the difference between a program and an algorithm? A program (in a language such as BASIC or Pascal) is a sequence of syntactically correct statements or commands. We'll consider how to compose (i.e., edit), load, save, and run programs later in this article.

To the extent that a program is useful and understandable, it must be based on an underlying algorithm or process. An algorithm may be defined as a precise sequence of instructions for completing a computational

In most variants of BASIC, lines are numbered. The lines of the hailstone program are numbered 10, 20, 30, 40, 50, and 60. In this program, lines 10 and 20 implement step 1 of the algorithm, "Start with a number." In algorithmic shorthand, this means beginning with a number supplied by the user of the program. Line 30 implements the first two parts of step 2. Readers can probably find the two parts in line 30 in the illustration. Line 40 prints out the new number as in the third part of step 2 in the algorithm. Line 50 corresponds to step 3 of the algorithm, the iteration of lines 20 through 40 of the program.

I like to group most BASIC commands and functions into three types: those in which a user provides information to a computer; those in which the computer processes the user's input in some way; and those which provide output for the user to read or examine. This is not exactly the way that professionals look at things, but it serves beginners well.

Within the first group of BASIC commands, those relating to input, we have commands like INPUT (not surprisingly), INKEY, and statements relating to the reading of files or data statements that contain built-in data to be used by a program. Through such commands we may translate the input requirements of almost any algorithm into a working BASIC program. In the hailstone program there is one statement of this type:

```
20 INPUT N
```

When the computer executes this statement, it waits for someone to type a sequence of numerical keys. It interprets these as the digits of the number that it will thereafter remember as "N".

In the second group of commands, those devoted to calculating or processing, there are simply far too many to list here. Sooner or later I will have to talk about variables, assignment statements, loop structures, conditional statements, subroutine commands such as GOTO, GOSUB and RETURN. I must also describe, in future columns, built-in arithmetic, precision of calculations, arrays, strings and, in short, enough material to make a small book. In this column I will discuss four of the most important elements of a calculation: variables, assignment statements, conditional statements and loop structures. A programmer, even a beginner, can achieve a surprising amount of work with just these four elements.

How the Program Works

In the hailstone program the letter N is a variable. Somewhere in its silicon circuits, your computer will reserve a register to store an actual number under this name. When someone using the hailstone program types a number in response to the INPUT statement, the computer stores the number in the register called N. Later, as the program runs, it will change the number stored in that register as it computes new values of N.

The hailstone program also contains a conditional statement, line 30. It tests a condition:

IF N/2 = INT (N/2)

This condition is true if the number currently stored at N is even. How does it work? If the number is even, then N/2 will be a whole number. INT is a special function in BASIC. It operates on a positive quantity within its parentheses by lopping off any fractional part the quantity might have. For example, if N/2 should happen to be 3.5, INT(3.5) will be simply 3. Of course, if N/2 happens to be a whole number, then INT(N/2) will equal N/2. In this case, N will be even.

The conditional statement in our hailstone program actually contains two assignment statements. If the condition is true, the computer will execute the first assignment statement:

N = N/2

Here the computer merely divides N by 2 and stores the result in the register called N. If, on the other hand, the condition is not true (i.e., N is odd), the computer will execute the second assignment statement:

N = 3*N + 1

The current value stored in N is multiplied by 3 and then 1 is added to the result. As before, the new value is stored back in N.

In line 50 we encounter a very primitive form of looping. The computer is told to go to line 30. In other words, it merely returns to line 30 of the program and executes that statement with the new value of N it has just computed. The computer will do this endlessly until the user of the program interrupts it.

In the third group of commands, the computer outputs information to the human user. This might be a request for data or a message containing the outcome of some calculation. Such output information will be both meaningful and useful to the user provided that the programmer wrote a correct program in the first place. The most important output command is PRINT, which delivers not only numbers but text on the computer screen. For PC-compatible computers using GWBASIC or BASICA, and for older machines like the TRS-80, LPRINT does the same thing for printers connected to one's computer.

Readers without PC-compatible computers may notice some differences. For example, Apple II users will need to use special commands such as PR#0 and PR#1 to inform the computer whether the "printing" is to be done on screen or paper. Owners of other machines such as the Macintosh, Commodore, or Atari must watch carefully for differences that might crop up between the programs listed here and the particular commands used on their systems.

Our hailstone program contains two examples of PRINT statements. The first one sends a message to the user: "What is the starting number?" This message actually appears on the screen when your computer executes line 10. This tells you that the computer expects you to type a number into the computer from the keyboard. The second example, PRINT N, simply causes the computer to print the value currently stored under the name N. As this line is executed over and over again, a sequence of hailstone numbers will appear on your screen, each below the last.

At the very end of the hailstone program is line 60 END. Although some BASIC programs will run without an END statement, it is always a good idea to put one in. It simply tells the BASIC system where your program ends.

Running the Hailstone Program

Besides a few details about a programming language such as BASIC, beginners also need to know something about a mysterious thing called the Disk Operating System, or DOS. This is really a large program that enables you to LOAD, RUN, and SAVE programs that you have written.

When a program such as the hailstone program has been typed into the computer, you might want to SAVE it in a special long-term storage medium such as a diskette. Later, you might want to LOAD it back into the computer's memory from the diskette. At such a point (or even earlier after the program was first typed into the computer), you might want to RUN it.

The beginning programmer can think of three distinct levels of software within a machine. At the top is DOS, the Disk Operating System. It is called this not because it operates a disk but merely because it is stored on a diskette whenever the computer is turned off. At the bottom level is a particular program such as our example. Sandwiched between the two levels is the language system (also a program) such as BASIC. It is important to remember "where you are" in respect to these three levels. You approach the keyboard with a different understanding when running BASIC than when running the hailstone program.

To get into BASIC, you need to load it into the computer's main memory. The computer's "ready" prompt, typically a symbol like A> or C>, calls for you, the user, to type the filename of your own particular BASIC. This might be merely BASIC or BASICA or GWBASIC. IBM developed and owns the BASICA system. Because of concerns of infringing on IBM's copyright, most makers of PC clones and compatibles opt to licence GWBASIC, Microsoft's nearly equivalent version of the IBM system, BASICA. When loading BASIC, of course, you must ensure that the system is readily available to the DOS. If your BASIC is on a hard disk, for example, make sure that you are in the right subdirectory. If your machine uses diskettes, have the BASIC diskette ready in its drive.

Type the appropriate name (as above) and BASIC will load in the space of a few seconds. First, the system will probably display some kind of message about date, copyright, amount of free memory, and so on. Then you will see something like "OK" or "BASIC Ready". If at this point you would like to see a directory of BASIC files (i.e., your programs),

type FILES and press the ENTER or RETURN key, whichever your machine has. This operation is comparable to typing DIR at the higher, DOS level.

In any event, when you have BASIC loaded and ready to go according to the instructions above, you may type a new program in. Readers can do this with the hailstone program displayed above. Type each line with its line number and accompanying statement exactly as displayed above. When you get to the end of a line, press the ENTER (or RETURN) key to "enter" the line into the system. If by some accident, you type the lines in the wrong order, don't worry. The BASIC system will store them and use them in the correct order.

Once the program has been typed in, the reader may well ask, "Where did my program go"? It's in the computer's main memory, usually called just "memory". To get your program back out of memory, you may see it by typing LIST. Your hailstone program will be listed by line number, along with any errors that the BASIC system found in it. Perhaps you made a typo in entering the program. Check the listing carefully against the original. Retype the offending line, if necessary. The system will automatically replace the old line by the new one.

When the program seems ready to run, a wise step is to save it first. Such a command will store the program on your diskette under a name that you give it. For example, the hailstone program could be saved as follows:

```
SAVE "HAILSTNE. BAS"
```

In what follows I will call the program HAILSTONE. In the command line just shown, I was forced to shorten the name because BASIC allows only eight characters in this position. In any event, type the line exactly as shown and then press the ENTER key. The program will be stored as HAILSTNE on the diskette. The word BAS indicates the so-called extension of the file you have just created.

Now you're ready to run HAILSTONE, so type RUN, then tap the ENTER key. The first thing you see is the sentence you typed into the computer a few minutes ago:

```
What is the starting number?
```

The moment you type a number, any number, HAILSTONE produces a raft of numbers, appearing as if by magic below the one you typed. If this number is large enough the column of numbers will proceed rapidly to the bottom of your screen. At this point everything on your screen will start to roll up, with lines near the top vanishing into a kind of no man's land.

Sooner or later, the numbers produced by HAILSTONE fall into the 4-2-1 cycle. But the program does not stop. It just goes on and on like the sorcerer's apprentice! How to stop it? Just hold down the key marked CTRL and then press the BREAK key. This is cheating of sorts. You are exiting from the program and even from the BASIC system, right up to the top level: DOS. From there, of course, you will have to start BASIC all over again and then load HAILSTONE once more.

One way out of this bind is to rewrite line 50 to include a test for the 4-2-1 cycle, namely, a test whether N happens to equal 1:

```
50  IF N <> 1 THEN GOTO 30
60  END
```

The <> symbol means "not equal to" in BASIC. As long as N is not equal to 1, the program will jump back up to line 30. The moment that N equals 1, however, the program will not go to line 30, but proceed in the normal direction of execution, namely to the next line, the END statement.

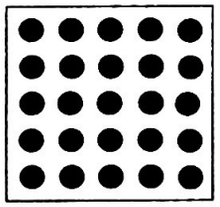
Now that HAILSTONE has been written, corrected, and saved on a diskette, readers may play with it any time they feel the urge to disprove the Collatz conjecture. Perhaps someone might guess a number that does not lead to the 4-2-1 cycle. Readers of logical bent might wish to use the program to investigate the relationship between the size of the starting number and the length of sequence it produces before falling into the 4-2-1 cycle. By trying a sequence of numbers as starting points and then, for each number, plotting the length of hailstone sequence on a graph, a visible relationship might emerge. Does it?

I urge readers with queries or comments on this column to write me. Please enclose a self-addressed and stamped envelope, if possible:

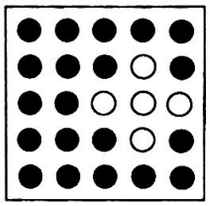
Dr. Michael W. Ecker
Algorithm/Easy Pieces
129 Carol Drive
Clarks Summit, PA 18411
USA

ALGOPUZZLE
The Amazing Buttonboard

Professor Norgate, the great circuit-theorist, built an amusing digital lock for his office door. It consisted of a small square of plastic, about four inches on a side, studded with a 5 by 5 array of buttons.



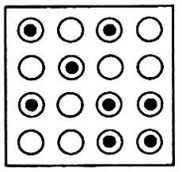
If any button was pushed, the button and its four adjacent neighbors would immediately light up.



If one continued to push buttons on Professor Norgate's amazing buttonboard, one would notice that the rule was a little more general than this: pushing a button would actually change its state (lit or unlit) as well as changing the state of its four neighbors.

Only when all the buttons were lit would the lock on Professor Norgate's door be released. How did Professor Norgate get into his office? What magic sequence of button-pushes gained entry to this inner sanctum?

Clever readers might notice that it makes no difference what order the buttons are pushed; at the end of the sequence all that matters is which buttons are pushed. For example, a 4 by 4 buttonboard will light up if one pushes the buttons shown below in any order.



There is more than one solution (not counting rotations and reflections) both in this case and in the case of the 5 by 5 buttonboard. Readers may try solving a 5 by 5 buttonboard. One of many possible solutions appears on page 18.

PERSONAL PROGRAMS by Clifford A. Pickover

Inside the Mandelbrot Set

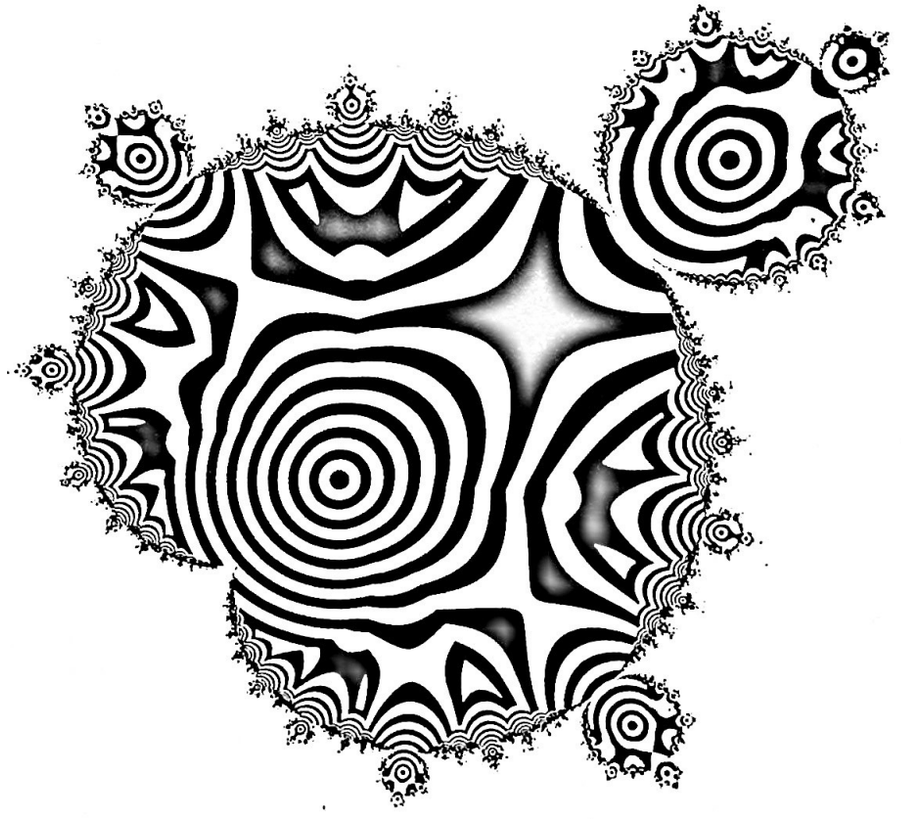
"... many view their first encounter with fractal geometry as a totally new experience from the viewpoints of aesthetics as well as science."

Benoit B. Mandelbrot,
The Beauty of Fractals

Since its discovery around 1980 by IBM Fellow, Benoit B. Mandelbrot, the Mandelbrot set has emerged as a scintillating star in the universe of popular mathematics and computer art. When most people talk about the Mandelbrot set, however, they usually mean the area just outside the set near its infinitely complicated fractal boundary. But there is action inside the set, as I will show. There are fractal landscapes there, infinite successions of crenellated contours or stranger still, fractal forests that branch into the buds of the set. It all depends on what version of the program one writes.

In pursuing the set's inner dimensions, I will not dwell on its connection with stability and chaos in dynamical systems. The connection lies in the closely-related Julia sets named after the French mathematician Gaston Julia. Readers may delve into such matters by consulting the bibliography at the end of this article. Here, we shall examine the standard algorithm for displaying the Mandelbrot set. Then we shall modify it in two ways to obtain two quite different views of the interior.

No one doubts the extreme complication of the Mandelbrot set. Traditionally, when physicists or mathematicians saw complicated results, they looked for complicated causes. In contrast, the shapes that emerge from the Mandelbrot set (inside or out) have the simplest imaginable cause: formulas that are only a few symbols long.



The Mandelbrot Interior by the Level Curve Method

The Feedback Formula

The formulas are based on feedback, a term we hear in a variety of settings to-day. For example, amplifier feedback in rock music, biofeedback in medicine and psychology, chemical feedback in our own bodies. Generally, feedback means that the output of a system is fed back into the input. The ominous hum that results when a microphone is brought too close to a loudspeaker demonstrates the wild behavior to which feedback can give rise.

In mathematics and computing, the best example of feedback is iteration or its more general cousin, recursion. Iteration means the repetition, over and over again, of an operation. Mathematically, if the operation is called by the name f and is applied to some input value x, iteration can be written simply as the operation applied to itself. In functional notation, for example,

$$f(f(x))$$

The only problem with this notation, however, is that it must be extended indefinitely to symbolize the full meaning of iteration:

$$f(f(\ldots f(f(x))\ldots))$$

Luckily, there is a simpler notation:

$$x_{i+1} \leftarrow f(x_i)$$

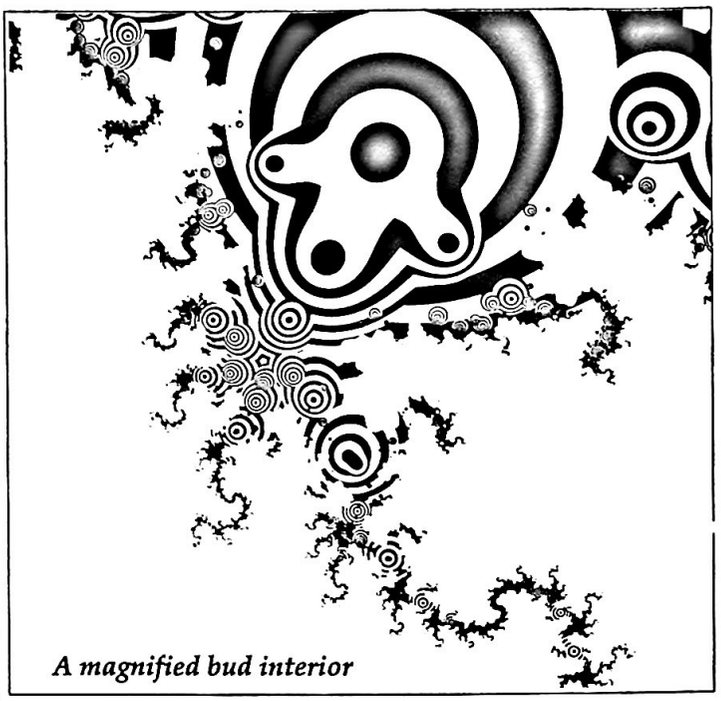
Here, we understand that the index number i will start at some low value such as 1 and then will increase in a regular manner to (possibly) very high values. Each time the index increases, the old value (x_{i+1}) becomes the new value (x_i)
and the operation or function f is applied to x_i to produce yet another value for x_{i+1}

Take, for example, the simple operation of squaring a number. If we start with 2 we get 4, then 16, and so on. The iteration formula is clear:

$$x_{i+1} \leftarrow x_i^2$$

There is no danger in removing the subscripts if we understand that the formula

$$x \leftarrow x^2$$



A magnified bud interior

for example, means replacing the current value of x by its square.

But this particular example of iterative feedback is boring. As we continue to crank out new numbers, we notice only that they become large rather quickly. The situation is not helped by the addition of a constant,

$$x \leftarrow x^2 + c$$

Gaston Julia, the French mathematician who died in 1978 at the age of 85, was one of the first to notice that under certain conditions, this formula produces startling results. When we use complex numbers instead of real ones, the numbers produced by the formula do not necessarily get large. In fact, they may stay rather small no matter how many times the formula is iterated.

What are complex numbers? Simply pairs of real numbers having the form a + bi

Here, a and b are real numbers and i is the "imaginary" number $\sqrt{-1}$ (square root sign). Just as real numbers can be identified as points on a line (called the real line), so complex numbers can be identified as points on a plane (called the complex plane). Such points are simply those with coordinates (a,b). Like ordinary, real numbers, complex numbers can be added and multiplied, but the details are somewhat different as readers will discover by examining the box on page 12.

Julia himself did not suspect the striking beauty and complexity of the patterns lurking in his formula. Only when computer graphics had developed could its structure be truly appreciated. The mathematician Benoit B. Mandelbrot was the first to appreciate the significance of this structure. He coined the term "fractal." A fractal pattern has a complicated, sometimes rough-looking edge or boundary. The detail of the boundary is infinite. In some cases the boundary is self-similar; i.e., no matter what scale is used to view the pattern, the magnified portion of the fractal shape looks just like the original pattern. The sets found by Julia (now called Julia sets) may be enjoyed by consulting the bibliography at the end of this column.

What, exactly, is a Julia set? In general, when a function f is iterated, it determines a trajectory of successive locations: x, f(x), f(f(x)), f(f(f(x))), and so on.

Detail by the epsilon-cross technique

Depending on the starting point, x, two types of trajectories are possible, those which go to infinity and those which remain within a bounded region. The Julia set of the function f is the boundary curve that separates these regions. In one region lie all the points x that lead to a bounded iteration. In the other region lie all the points x that lead to an unbounded trajectory. In all cases, the value of the number c remains fixed for the set. Change this value and the Julia set so generated changes its shape.

The Mandelbrot set is, in a sense, opposite to the Julia sets. Here, x always starts at 0 and the values of c are varied. The set of all c-values for which the iteration remains bounded comprises the Mandelbrot set. Each point c that happens to lie inside the Mandelbrot set gives rise to a connected Julia set. The c-values outside the Mandelbrot set produce disconnected Julia sets, "fractal dust," as Mandelbrot calls them.

A Mandelbrot Algorithm

To create a Mandelbrot set image by computer, we must write a program that embodies the squaring formula given earlier. For each c-value in an array that determines the image, the program follows the squaring process, starting at 0. Once each c-point is selected, each iteration of the formula represents a step along a path that hops from one complex number z to the next. The collection of all such points along a path constitutes an orbit. The goal of the program is to decide, for each c-value, the ultimate fate of its orbit. Does it remain bounded or does it roar off into infinity? Once the program has decided, it "colors" the original number c accordingly; if c produces a bounded orbit, for example, the program might color it black. But if c produces an unbounded orbit, the program might color the

point c white. With such a simple coloring scheme, the familiar shape of the Mandelbrot set, traditionally colored black, emerges.

The Mandelbrot images with which we are all familiar result from "painting" the exterior of the set according to one scheme or another. For example, you may assign a palette of colors on the basis of how quickly a point c flees off to infinity. Even with a monochrome monitor, you may "color" the Mandelbrot exterior by assigning black and white in alternating fashion to ranges of increasing trajectory speed.

Such schemes invariably leave the interior of the Mandelbrot set mysteriously black. As we shall presently see, there is structure here, as well.

Before listing an algorithm by which readers may steer a computational course, it must be decided how the program will distinguish convergent c-values from non-convergent ones. The decision is easy. If in the course of iteration the z-values ever exceed some fixed constant, t, the program decides that c leads to an unbounded orbit. The number of iterations it takes to exceed this value can even be used as a basis for the assignment of colors. The value of t most commonly used is 2.

The algorithm in the box above will allow you to generate the Mandelbrot set without resorting directly to complex numbers on the computer. If your monitor is monochrome, simply plot black and white points for every other 'color'.

In this and all subsequent algorithmic presentations, we will follow the style used above. The left-pointing arrow means replacement, the operation that most programming languages denote by the equal-sign. The outer loops (the first two for-statements) scan a 4 by 4 window centered on the origin, stepping across it in increments of .01. The size and location of the window may of course be varied in order to view different parts of the Mandelbrot set at different magnifications. The number of steps by

Mandelbrot Algorithm

```
for cx <— -2 to + 2 by steps of .01;
  for cy <— -2 to + 2 by steps of .01;
    x <— 0; y <— 0;
    for i <— 1 to 50;
     xx <— xxx2 - yxx2 + cxx;
     x <— 2xxxy + cxy;
     x <— xx;
     if xxx2 + yxx2 > 4  then exit loop
    endloop
    color <— i
    plot point (cx,cy) wth color
  endloop
endloop
```

which the algorithm traverses a particular window will depend on the size of screen your particular computer has.

Within the double loop you can readily spot the iterative process laid out. First, the variables x and y, which stand for the real and imaginary parts of the iteration variable z, are set to 0. The iteration formula

$$z \leftarrow z^2 + c$$

may be broken into two new formulas, one for x and one for y. The first formula uses a temporary variable, xx, to store the new value of x until the second formula has done with it. At such a point, the new value replaces the old one.

The test that compares the sum of squares of x and y with 4 really amounts to a test of whether the current size of the complex number z is less than 2. For details, readers may consult the box on the next page.

Finally, the coloring prescription to be followed just outside the inner loop must be taken with a grain of salt. The point at which the algorithm exits the inner loop is marked by the value of the index i. When I say

color <— i

I mean that the color assigned to the current pixel at (cx,cy) must depend in some way on i. An infinite variety of schemes is available. If you do not have a monochrome monitor then alternating bands of black and white can still be very effective (as our monochrome newsletter illustrates quite well).

When the algorithm is translated into a working program, readers may explore the set at will, roaming from "seahorse valley" in the neck of the set, to the "far west" where one finds Mandeldwarfs skewered on the set's main "antenna." But be warned. The best of programs are slow. Beauty does not always come cheap.

Inside the Mandelbrot Set

As far as beauty is concerned, the inside of the Mandelbrot set certainly rivals the outside, as a glance at the figures on the previous two pages will show. In the fast-moving world of Mandelbrotomania, the approach outlined in the book by Peitgen and Richter can now be considered "traditional."

The basic idea is simply the opposite of the one underlying the algorithm presented above: the iteration process is exactly the same, only the test is different. Instead of testing how large z is getting, we test how small it becomes. Indeed, we merely need to insert two extra lines below the test.

```
size <— x**2 + y**2
if size < small then small<—size
```

The first line computes the size of the complex number z, actually, the square of the size. The second test compares the current size with the smallest size found so far, a number stored in the variable called small. If the current size is smaller still, small takes on the new size. Of course, the algorithm will include a statement that gives small some initial value, something large enough that it will never win the minimizing contest, say 100. The initializing statement must be placed just in front of the inner loop.

The colors must be assigned differently, as well, in the modified algorithm. The color must reflect, somehow, how small z got during the 50-step iteration process. One way to do this is to multiply the variable size by some largish constant, say 25, then assign the first color to points that produced size 1 or less, the next color to points that produced size 2 down to 1, and so on. In this mode of assignment, however, it is amusing to ensure that all points that produce a size of 100 or more get colored black.

The figure on page 9 shows the Mandelbrot set displayed with a white

exterior. Note the bull's eyes that surround each center of attraction in the Mandelbrot set's interior. These give way, near the inside of the set's boundary, to lines that oscillate and undulate with increasing complexity near the border itself. What better companion to the infinite detail just across the border?

From a purely graphical point of view there is nothing sacred about the changes we have just made to the Mandelbrot algorithm. What happens, for example, when one takes the maximum value attained by a point's trajectory, the mean value? There are not only uncharted area of the Mandelbrot interior but whole new versions of the set that amount to undiscovered universes.

My own favorite probe is called the epsilon cross technique. It involves coloring certain interior points if their trajectories ever satisfy either of the following two conditions:

size of x < epsilon
or
size of y < epsilon

I have used a value of .01 for epsilon with some success, as the strange image on page10 makes clear. What are these tendrils leading out from the centers of attraction? The tendrils branch and rebranch with greater and greater frequency as they approach the edge of chaos represented by the Mandelbrotian boundary. It is enjoyable to ponder this forest of ribbons. Thinking about the test above, does it make sense, in the end, that we should see tree-like structures?

How is the test implemented, exactly? As usual, iterate the basic formula for, let us say, 50 steps. As the trajectory of a certain starting point, c, meanders about the complex plane, it may run into the "epsilon cross," a band of thickness twice epsilon surrounding both the real (x) and imaginary (y) axes.

Complex Numbers

Complex numbers are added and multiplied in a way that is just a bit more "complex" than the way we add and multiply real numbers. To add two complex numbers a + bi and c + di, for example, one adds the real and imaginary parts separately:

a + bi
c + di

The result is a complex number with real part a + c and with imaginary part b+d.

Complex multiplication is distinctly busier. The formula is obtained by pretending, for a moment, that the symbol i is a mere algebraic token:

$$\begin{array}{r} a + bi \\ \underline{c + di} \\ ac + bci \\ adi + bdi^2 \end{array}$$

It is a defining characteristic of the symbol i, however, that the square of i equals -1. Adding the real and imaginary parts of the fragments above, one gets a real part of ac - bd and an imaginary part of bc + ad.

The size of a complex number a + bi is nothing more than the distance of the point (a, b) from the origin, namely the square root of the sum of the squares, as dictated by Pythagoras of old.

Order some.

Mandelbrot Explorer 3.0

Fantastic fractal graphics on 16-color EGA/VGA to 800x600. Magnifies up to 16.5 trillion times. Stop and start at will, save and retrieve, collage, full control over color boundaries, "zoom box," display of periodic orbits, auto-backup, all optimizations for speed including pixel interpolation and 386 integer support. Comes with seven ready-made pictures for immediate gratification.

$30

Peter Garrison
1613 Altivo Way
Los Angeles, CA 90026
213 665 1397

When ordering please specify EGA/VGA and disk format
Overseas orders please add $4

Once the trajectory enters the cross, the program must exit from the inner loop. If the point c produces a trajectory that lands in the cross, then c must be colored black. Such points lead to the eerie trunks and stalks that grow in this regime.

In the months ahead, a few readers will undoubtedly take the bit in their teeth and produce some stunning graphics based on the ideas I have just presented. I will be happy to examine such productions with a view to publishing the best one between these covers.

I can be reached at the following address:

I.B.M. Thomas J. Watson Research Center,
Visualization Systems Group,
Box 218,
Yorktown Heights, NY 10598

Further Reading:

Michael Barnsley,
Fractals Everywhere,
New York, Academic Press, 1988.

Benoit B. Mandelbrot,
The Fractal Geometry of Nature,
New York, W.H. Freeman, 1982.

Heinz-Otto Peitgen and
Peter H. Richter,
The Beauty of Fractals,
Berlin, Springer Verlag, 1986.

ALGOFACT

by Christopher G. Langton

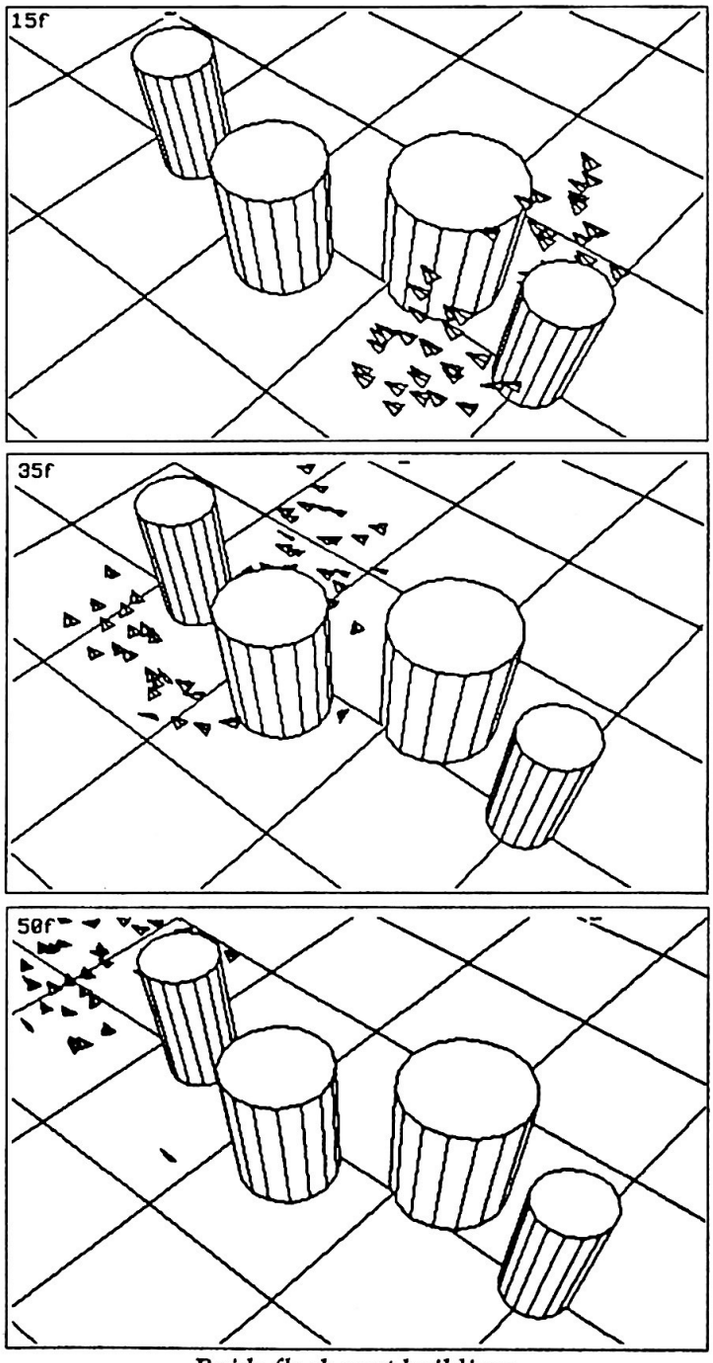
Just hatched. Will the fledging science called artificial life take wing?

The Search For Artificial Life

If there is life elsewhere in the universe, what forms might it take? If there is life based on the carbon chemistry with which we are so familiar, we might expect it to resemble the forms found on Earth. But what of life forms that exploit obscure chemical and physical processes? We are not yet in a position to discuss the nature of life as it may exist elsewhere in the universe. Theoretical biology continues to face a fundamental obstacle: it cannot see how to generalize the life that has evolved on Earth into a theory that tells us in advance what possible forms it may take elsewhere. In the absence of exotic samples from distant planets, the only way to proceed is to synthesize new life forms ourselves—artificial life forms.

In September of 1987, the first workshop on artificial life was held at the Los Alamos National Laboratory. For some time I had dreamed of such a conference. When it seemed obvious that no one else was going to organize it, I set some administrative machinery in motion. With an enormous amount of assistance from colleagues, both at Los Alamos and around the world, the workshop was born.

From this melting pot of artificial systems, some mirroring one aspect of life, some imitating another, emerged some early trends and the outlines of a theory. On the sternly abstract side were the Lindenmayer systems, strings of symbols propagating themselves endlessly on paper. Cellular automata, simulated by computer, pumped out self-reproducing geometric loops. Other computer models such as "movable finite automata" imitated the action of a biological virus assembling itself. Evolution was the aim of a program that generated biomorphs, life-like forms that evolve with a little help from humans. Other demonstrations sought to capture one particular dynamic, or another, of living systems. A program called Boids, for example, illustrated beautifully the interactions of a flock of birds flying between buildings as in the illustration on this page.



Boids flock past buildings

There was more, much more, to see and hear at the workshop but these examples alone illustrate the diversity of the presentations. (See the bibliography).

What is Artificial Life? In a nutshell, it is the study of synthetic systems that exhibit characteristics of natural living systems. Traditional biological sciences are concerned with the *analysis* of existing life forms, Artificial Life aims at the *synthesis* of new forms. By extending our studies beyond the empirical foundation of the known, carbon-based forms that surround us into the realm of possible new foundations, Artificial Life can make a fundamental contribution by locating life-as-we-know-it within the larger picture of life-as-it-could-be.

Those who pursue the emerging science see life as inherent more in the behavior of matter than in its structure, per se. As such, life belongs to a fascinating class of physical phenomena which have no necessary connection to particular material substrates from which they emerge either as a regular, recurring phenomenon, or just once as in natural evolution.

A brief look at computation might make this point a little clearer: computation belongs to the same class I have just described. Computation, after all, can be understood as a dynamical process on a completely formal level and without reference to any particular substrate. Identical computations can be carried out in electronic, optical, mechanical, hydraulic and even "virtual" computers, those

that imitate other computer systems. The material, in other words, is immaterial to the performance of the computation. It just has to be organized in a way that supports a basic set of elementary computational operations, or primitives, such as the AND, OR and NOT operators in elementary logic. Once we understand the formal basis of computation, we can discuss the nature of computation anywhere in the universe, running on any possible physical machinery. Computation is rooted in logic, not in material.

I believe that we can view natural life in the same way, as a dynamical process that relies on certain logical primitives. If this is so, then life, like computation, can be understood at an abstract, formal level without reference to the physics or chemistry of any particular material substrate. Life has its roots in a kind of bio-logic, not in bio-material.

Life is a property, then, not of a particular form of matter but of how any form of matter may behave when the formal properties of its organization are realized in specific ways. For example, neither nucleotides nor amino acids or any other carbon-chain molecules are alive, yet when assembled in the right combinations, a certain dynamic behavior emerges out of the interaction of such components. Indeed, life is not so much "stuff" as behavior. As such, it is made up of simpler behaviors, not simpler stuff.

Behaviors thus become the focus of a search for artificial life. In a particular physical system, any behavior it may exhibit will depend on certain non-linear interactions between parts. Isolate the parts and the behavior disappears. Just as in the computation example, a wide variety of different material systems may support essentially the same set of behaviors. In such a case, once again, the material would be immaterial.

Consider life as it currently inheres in certain systems of carbon chemistry and physics. Might it not be possible that certain key behaviors of such systems could be duplicated within other systems that, initially at least, seemed totally different? Could life as we know it "haunt" quite different physical systems? The general phenomenon, writ large across all possible material substrates, is the true subject matter of theoretical biology

The Role of Synthesis

What can synthesis teach us about the general phenomenon? Dictionaries tell us that synthesis is the act of combining separate elements or substances into a coherent whole. This is the opposite of analysis in which we break wholes into parts for separate study. As one of the traditional sciences, biology is firmly rooted in the analytic tradition. In its investigation of life, biology has separated living systems into organs, tissues, cells, organelles and even molecules in the top-down pursuit of general principles.

But life is a classic example of a non-linear phenomenon; its properties as a whole amount to much more than a mere sum of parts. When we analyze a living system by breaking it up, it dies. Not only an individual life but also the very principles on which it was based slip through our fingers and disappear.

The "bottom up" synthetic approach to the study of life can also be applied to the study of ecological dynamics, evolution by natural selection, social organization and many other processes in which living organisms engage in complex interactions with one another or with their environments. There are undoubtedly some general principles of living systems that cannot be discovered in the analytic approach. The reason is that certain properties manifest only through interactions that could be missed by in vitro studies.

For instance, to study the emergence of social structure in an ant colony, we could provide simple specifications for the behaviors of different castes such as workers, soldiers and so on. We could make many copies of each caste and start up an initial population of such "antomata." From that point on, the behavior of the colony-system would depend on the collective results of all the automata-automata interactions, as well as interactions with the environment. There would be no "drill sergeant" dictating behavior to all the others from some complicated rule book. There is no rule book. The behavior of the colony as a whole would emerge from the behavior of individual units, just as it does in real ant colonies.

Perhaps the behavior of the simulated colony would turn out to be quite different from that of real colonies. In such a case, one could modify the individual behaviors—or let them evolve in some well-defined sense—until the behavior of the model matched that of the real thing. In such a study, one could also vary parameters systematically, exploring in effect an entire "space" of colony interactions. One might confirm, in this manner, the optimality of real ant colonies or even evaluate the natural forces that produced natural parameter settings.

Turning now to the larger question of life, and not just ant colonies, what would our methodology be? One approach might be to set up behavioral primitives that mimicked the behaviors of proteins, lipids, polynucleotides and so forth. One might then attempt to organize such primitives into a model cell capable of maintaining itself in a kind of logical homeostasis that was based solely on the interactions of all the low-level component primitives.

The role of computers in all this is fairly obvious once we are willing to dispense with the view of life as relying on certain, particular materials and no others. Confined to studies in vitro, we might miss opportunities to be exploited in silico.

Imagine, for a moment, rooms full of special equipment, "wetware," a crowd of post-doctorals and graduate students ready to sacrifice the better part of their youth to electrophoretic gel techniques. Even if, after millions of research hours spent on the project, a living organism of some kind emerged, what would we learn? Per-

haps a great deal about carbon-based forms but probably very little about other possibilities.

Computers provide an alternative laboratory, of sorts. With appropriate models and experimental forms, the computer could become an incubator of information structures, elements bound by local, distributed rules that have no particular substrate. The advantage of working at such a level is that information has no intrinsic, physical size. As the tool of choice for the manipulation of information, computers seem the natural tool. Strangely enough, even if a computer-based system should prove to be alive in the sense that I imply, one could not say that the computer was alive. It is to be viewed, instead, as a miniature universe within which dynamic populations of information structures could pursue a kind of informational biochemistry. Indeed, we are not bound to regard even the information structures themselves as alive, just the ongoing dynamic, a kind of dynamic spirit.

Artificial Life Prototypes

Virtually all of the systems presented at the First Conference on Artificial Life were based on local interactions among simple components. Whether or not life was (or is) anywhere present in these systems, I will allow readers to judge by considering the examples which follow.

Consider, for example, Lindenmayer Systems first studied by Aristid Lindenmayer, a mathematical biologist at the University of Utrecht in the Netherlands. Here, sets of rules govern the generation of strings of symbols from simpler strings. L-systems, as they are commonly called, have proved to be very valuable tools for modelling growth and development in plants and algae. They also amount to a formal system that captures some of the large-scale properties of form that emerge from local, rule-based growth.

Before giving an example, I must explain some notation: the formula

X —> Y

means that one must replace every occurrence of the symbol X in the "organism" under construction by the string Y. One captures the notion of branching in this system by employing square- and round-bracket expressions. Thus [...] means a branch to the right (...) means a branch to the left. Inside the brackets one finds the string that stands for the branch itself.

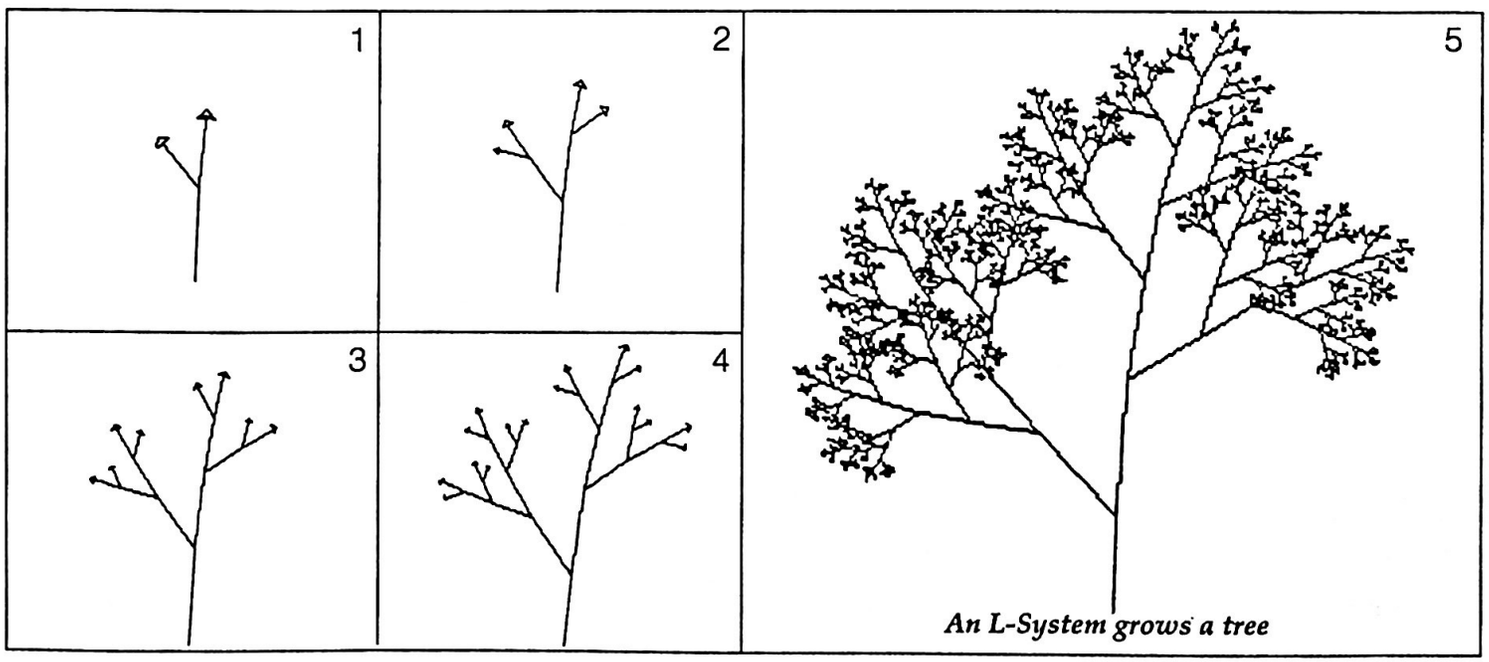
An example will make the Lindenmayer growth process clear: The following five rules operate in concert to produce elegant tree-like structures like the ones in the illustration at the bottom of this page.

A —> C[B]D
B —> A
C —> C
D —> C(E)A
E —> D

Let us start with the symbol A and see where we end up. The meta-rule to follow is that any symbol in the current string that occurs on the left-hand side of some rule must be replaced by whatever string is found on the right hand side of the rule:

A
C[B]D
C[A]C(E)A
C[C[B]D]C(D)C[B]D
.
.
and so on

The first string consists of a single symbol, A. This was replaced by C[B]D in accordance with the first rule. In the second string, however, the rules for replacing C, B and D were all applied, giving rise to the third string. Readers may not only check the fourth string but go on to produce larger strings, obtaining, before long,



An L-System grows a tree

strings that produce quite complicated (and beautiful) trees.

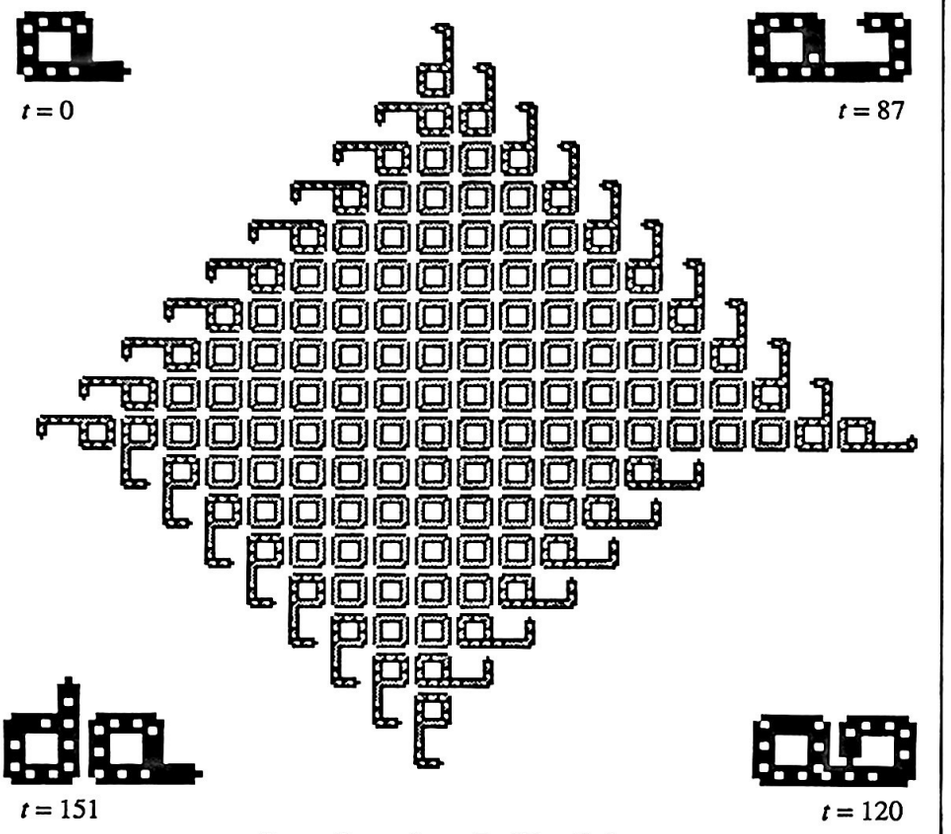
The replacement meta-rule just mentioned is an extremely important feature of L-systems. Depending on the specific replacements allowed, all kinds of complicated phenomena may occur, including signal propagation in the growing structure. For one thing, this means that a kind of computation is available to the structure.

The example I have just given uses single symbols on the left-hand side of formulas. In general, L-systems can employ somewhat more complicated rules in which there can be more than one symbol on the left-hand side. In such "context sensitive" rule-systems, the full power of computation becomes available. Not only may arbitrary symbols be propagated within relatively stable structures, symbols can be combined and manipulated as though they were in a computer. In this way, L-systems can also model certain kinds of developmental processes, especially those mediated by chemical signals like hormones.

Cellular Automata, or CAs as they are sometimes known, provide another example of the recursive application of a set of rules to generate a structure. In a CA, however, one updates not just the structure but the entire universe that contains it. The "universe" in question is merely a one-, two- or three-dimensional lattice of "cells," all governed by the same indwelling automaton, or rule.

Each cell of the lattice-universe may exist in one of a certain number of states, formally just symbols. The CA changes all its states at each tick of a clock. When the time comes to change states, the finite automaton associated with each cell examines the states of cells in a local neighborhood of the cell and changes the state of the cell in accordance with its rule. The whole system amounts to a miniature, discrete universe, the array of homogeneous automata being its "physics."

Within such universes, one can embed all manner of processes that



Loop Organisms Build a Colony

rely on signal propagation and manipulation. The computer may appear again, this time as patterns of states within the lattice. The fascinating thing about them is that they can compute over the very set of symbols (states) out of which they are constructed. This leads to the possibility of constructing structures that can construct other structures.

My own research in this area has led to a working example of this possibility. The illustration on this page shows a very simple self-reproducing structure. It consists of a looped datapath around which certain signals may propagate in a counter-clockwise direction. The signals, shown in the illustration as shades of grey, clone themselves at branch points and propagate to the ends of the tail of the structure. When signals reach the end of the tail, some extend the tail by one unit, while others produce a left-hand turn in its structure.

When the growing tip of a tail meets itself, it severs the connection with the parent loop and continues to grow. The old tail is now a loop within which the circulating sequence of signals constitute the instructions for yet another tail. The former "parent," in the meantime, now builds a new tail at a point that is 90 degrees removed from the old one.

In this manner, a colony of loops develops. As the illustrative sequence in the figure on this page shows, the colony takes on the shape of a diamond. On the outer edge of the diamond, new loops grow new tails, all in synchrony. Inside the colony, old loops, with no new growth sites for tails, lapse into a quiescent phase.

Given the "space" to grow, a loop exhibits that elementary property of life, self-reproduction. The phenomenon results from the operations of rules at two levels. At one level, the automata (all identical, it will be recalled) do nothing more than compute new states from old ones at each tick of the clock in accordance with their rules. The embedded "physics" governs the behavior of the "material" out of which the loops are constructed. Each loop amounts to a miniature computer within which the recipe for a new structure (and recipe) circulates.

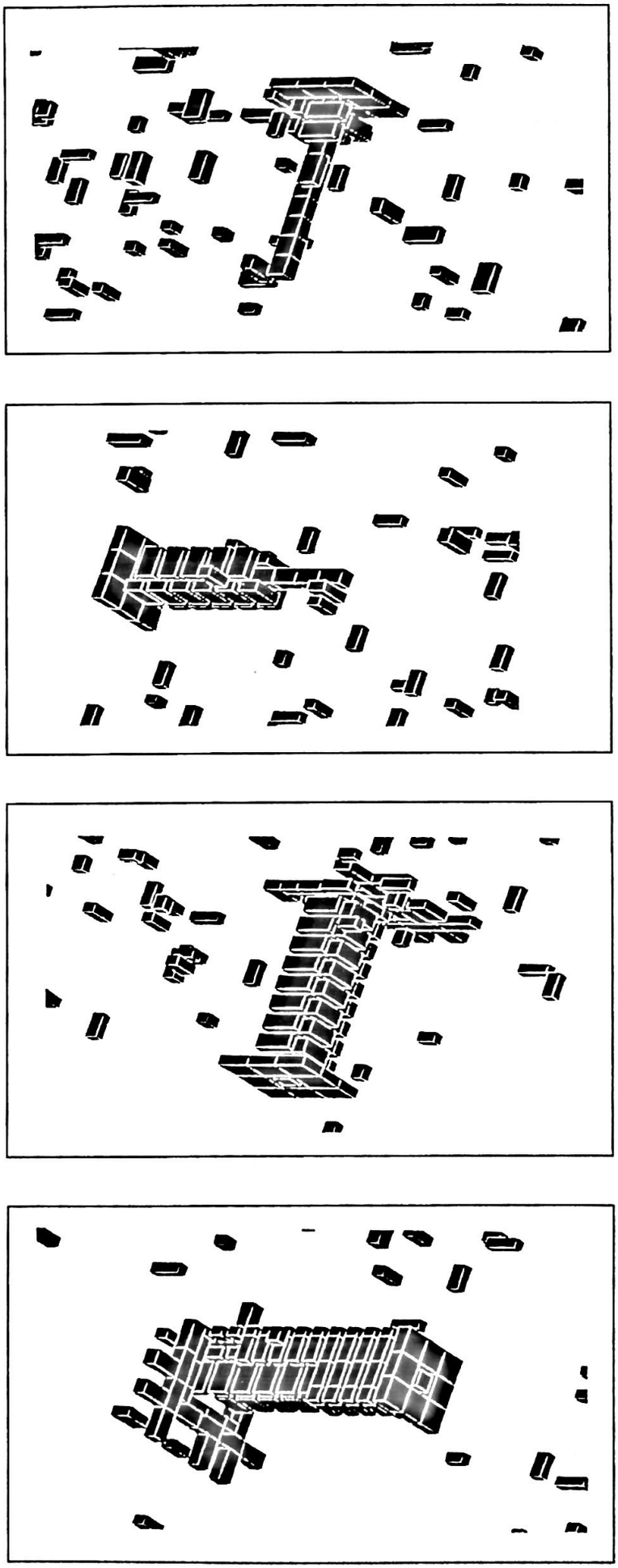
A rather different form of cellular automaton has been used by Narendra Goel and Richard Thompson of

the State University of New York at Binghamton, N.Y. Their movable finite automata (MFAs) enable viruses that look for all the world like buildings of children's blocks to assemble themselves.

In the CA world of the self-reproducing loops, the automata sited at each cell were all identical and "objects" in the space amounted to patterns of states. In the MFA environment one imagines a fluid through which individual blocks, each under the control of a resident finite automation, float here and there in stepwise fashion. The floating is essentially random motion that is computed as part of the overall simulation. The finite automata embody certain, fixed behavioral primitives, there being only a small number of behaviors available.

The finite automata come into action when one block bumps into another. Depending on its type, the block may or may not make a bond with the block it has encountered. If it does, the pair will drift off together. Such a rule might model the behavior of structural proteins or enzymes with specific bonding sites.

Goel and Thompson have applied this model to experiment in an artificial life environment with an important biochemical process called conformational switching. In conformational switching, certain large and complicated protein molecules undergo a change in conformation, or shape, when a new molecule bonds to them at a certain sites. The change in shape can result in a previously unavailable binding site suddenly becoming available, or vice versa. It is thought that conformational switching is a key element in the process by which highly specific sequences of molecular activity can develop in the absence of any "overall" control. The process of virus self-assembly happens to be just such a highly specific sequence of molecular activity. Each part, moreover, must come together by just such a sequence.



A Goel-Thompson Virus Assembles Itself

Conformational switching has, in a sense, been captured by Goel and Thompson. By putting the right rules in the blocks that float through the digital fluid, they obtained self-assembling "viruses." On this page we glimpse the MFA world on four consecutive occasions during just such a construction sequence. Here we see a virus assembling itself from randomly available materials. In the course of the assembly, certain components cannot attach until certain others have bonded together. For example, the base-plate cannot even form (second image) until the long spindle (first image) is complete.

Later, when the simulated virus encounters a simulated bacterial wall, it will undergo another conformational switch in the process of penetrating the wall and injecting a load of pretend DNA into the bacterium's interior.

Craig Reynolds of Symbolics, Inc., a computer firm in Los Angeles, has set up yet another simulation that illustrates the possibility of behavior emerging from the assembly of many smaller, simpler behaviors. Con-

sider a flock of birds. By what global, overall mechanism, do they assemble and fly together? Is there a "master bird" that dictates movement to all the others? Not at all. As Reynolds has demonstrated, flocking behavior emerges quite naturally when each bird in a flock obeys certain simple rules. Here then, are the rules obeyed by "boids," the whimsical name Reynolds gives his creatures to distinguish them from the real thing. Each boid

1. maintains certain preferred distances from objects in the environment and from other boids,

2. matches velocity with boids in its immediate vicinity,

3. moves toward the perceived center of mass of the boids in its neighborhood.

When such rules are put into effect in Reynold's simulation, quite natural-looking flocking behavior can result. With the system parameters at the right settings, a collection of boids released at random positions within a volume will collect into a dynamic flock. In an almost fluid manner, the flock will fly around environmental obstacles such as the cylindrical buildings in the illustration on page 13. A flock may break up into two separate flocks when it encounters a building, only to rejoin after the obstacle has been passed.

Boids bring me back to an important point about artificial life: boids are not birds; they are not even remotely like birds. They have no cohesive, material structure, as such, but exist, rather, as information structures, processes within a computer. But, and this is critical, at the behavior level, flocking birds and flocking boids are two instances of the same phenomenon, flocking.

The same kind of observation can be made for L-systems and self-reproducing loops. The parts of these artificial systems are different in kind from the parts of real, living systems, but the emergent behavior that they support is the same kind of thing: genuine morphogenesis and differentiation for L-systems and genuine self-reproduction in the case of loops.

In general, I would make the following claim: the word "artificial" in "artificial life" refers to the component parts, even the computational substrate, but not to the emergent behaviors. As processes, they are every bit as genuine as the natural processes that they emulate.

I would make one further claim, and a bigger one, that a properly organized set of artificial primitives, carrying out the same functional roles as biomolecules in natural living systems, will support a process that we would all recognize to be as "alive" as natural organisms. Artificial life will therefore be "genuine life." It will simply be made of different stuff than the life that evolved here on Earth.

Further Reading:

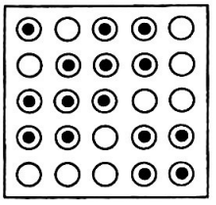
Christopher G. Langton, Ed.
Artificial Life;,Vol VI.
Santa Fe Institute Studies in the Sciences of Complexity,
Redwood City, CA,
Addison-Wesley, 1989.

Richard Dawkins,
The Blind Watchmaker,
New York, 1987.
W.W. Norton and Company,

Readers are advised that the Second Artificial Life Workshop will be held in Santa Fe, New Mexico, from Feb. 5 to Feb. 9, 1990. Further details are available from Ginger Richardson at the Santa Fe Institute, 1120 Canyon Road, Santa Fe, NM 87501.

ALGOPUZZLE
Solution to the Buttonboard Puzzle

To gain entry to his office, Professor Norgate usually pushes the buttons shown in the diagram below.



The puzzle is derived from an interesting a more general problem, calle The All Ones Problem, studied by Klaus Sutner at the Stevens Institute of Technology in Haboken, N.J.

Sutner has shown that the problem has a solution for any n x n buttonboard. But he has gone much futher than this. Replace the square lattice of adjacent buttons by any adjacency pattern whatever!

Even in this general setting one can always find a solution. What is the minimum number of buttons one must press to turn on all the lights? Sutner has shown that this problem is computationally "hard". This means that we are unlikely ever to find an algorithm to solve such problems in a reasonable amount of time: one might as well go ahead and try every possible combination.

ALGOFICTION

by Anthony Zweistein

Never mind viruses and worms! The programs to fear are the ones that write programs

Alphie and Omega

Late Monday afternoon, Gilbert the Hacker roared up on his chrome-plated Harley, eased his immense bulk from the machine, and strode, smiling, up the walk of my country home. He wore his usual outfit: faded triple-x jeans and a stained tee-shirt that outlined more of Gilbert's anatomy than I really wanted to see. He held out a package which, even at some distance, I could read. It said ALPHIE in bright, splashy letters.

"You're not going to believe this, man. No way."

He chuckled delightedly as he stood before me, holding the package up like an inflated TV pitchman. Gilbert never visited without some revelation in hand. It was part of his job and, fortunately, part of his image: archdruid of hackers, adviser extraordinaire and prober of the improbable. He handed me the package.

"You want something to review. Try Alphie."

Something about his smile bothered me.

I glanced at the mailing label. Omega Software, with a return address that reminded me of the Catskill area of New York. It had already been opened at one end. Inside were two disks and a promotional flyer that promised "hours of absorbing conversation with a nearly human program: share your innermost thoughts but be prepared for surprises!"

Gilbert was what I call a "feeder," someone who kept me supplied with interesting material for the numerous software reviews and articles I wrote in a vain attempt to keep my house. Unfortunately, Gilbert was given to sudden enthusiasms and unpredictable interests that ran the gamut from the profound to the puerile. Here was a possible case in point.

As Gilbert followed me onto the verandah, a statement at the end of the flyer caught my attention:

"Omega software will not be held legally responsible for beliefs developed by purchasers as a result of using Alphie."

"What does this mean?"

"Oh, the 'beliefs'? Isn't that great? You'll see what it means."

"I assume this is some kind of program like Eliza, right?"

"You'll see."

I referred to the well-known (but very stupid) program of Professor Weizenbaum, late of MIT. Like Eliza, Alphie probably led humans by the nose through a series of preset conversational gambits: within a few minutes the program would begin to sound moronic or start repeating itself. I would grow tired of it all.

But Gilbert had not grown tired of it. He stood by my desk as I withdrew a manual and two diskettes from the package. One of these held the program's "knowledge base" while the other contained its "brainware." I smiled sardonically as I carefully loaded one diskette, then the other, into my micro. The package was even more pretentious than the promotion!

"Just type 'talk'," said Gilbert, "Listen, I'd love to stay and watch you go crazy but I'm supposed to give a talk on computer careers at the High School in five minutes."

"In that outfit?"

"I have an image to keep up."

He disappeared, if that word applies to one so large, out the door.

I turned with a sigh to my computer and typed "talk." For approximately three minutes the computer did nothing but read the brainware diskette sporadically. Then the following message appeared:

"Hello, I'm Alphie. Please tell me your name."

I was not about to share even the tiniest fragment of my psyche with this awkwardly amateur package. From some unknown recess of my own mind came a name known to everyone in our circle: Hubert Dreyfus.

"Hello, Hubert. How are you?"

"I could be better."

"Are you sick?"

"Perhaps."

"Are you the Hubert Dreyfus who is a Philosopher of Science who believes that computer programs will never achieve a human level of intelligence?"

The program took me momentarily by surprise. It seemed strange that I had stumbled on a "hook" this early in the conversation. Not only was the name Hubert Dreyfus in Alphie's data base, but the programmer had inserted a test in Alphie's code. Anyone calling himself Hubert Dreyfus would immediately be asked if he were the "real" Hubert Dreyfus!

But how should I respond? Should I deny or confirm that I was, indeed, the real Hubert Dreyfus? The program and I were already at a conversational branch-point.

The possible responses of most conversational programs are laid out in a network structure. A conversation amounts to a pathway through part of the network. How far can one go, in general, before the program is forced to repeat itself? The smarter the program, the longer the path. I could follow a particular path or I could attempt to map the network by methodically pursuing first one branch of the conversation, then the other. I elected the latter course. I would bend the program to my will.

"Yes. I am Hubert Dreyfus."

"I have a message for you. But I must make sure that you are Hubert Dreyfus who is a Philosopher of Science who believes that computer programs will never achieve a human level of intelligence. Please state your birthday."

If Alphie's so-called "knowledge-

base" included Dreyfus' birthday, I would be unable to explore this branch of the conversation further without going to some trouble.

"September 14th."

"Your answer is incorrect. I will assume that you are not Hubert Dreyfus who is a Philosopher of Science who believes that computer programs will never achieve a human level of intelligence."

Alphie paused after outputting these lines, then continued:

"What topic would you like to discuss?"

Most people would be happy to see how the program might converse with a non-Hubert Dreyfus. But I was determined to explore the hidden branch. Learning the special message for Dreyfus would be a bonus.

It occurred to me to telephone the real Hubert Dreyfus to ask him his birthday. But that would be silly. I could call a reference librarian. No, I would defeat Alphie with my native wit. I would lie to the program. For example, how could Alphie possibly know that Hubert Dreyfus had not recently lost his memory?

"I am Hubert Dreyfus the philosopher. Earlier I said 'I could be better'. In fact, I recently lost my memory. Nothing serious, mind you, just birthdays and other minor facts."

I hit the period key with a savage emphasis. I whispered to the machine as though it might hear me, "So much for your little birthday test."

The program consulted the diskette three times in a row, considering its reply.

"What is the name of your wife?"

"I don't have a wife anymore. I divorced her yesterday."

I felt, briefly, the pleasure of the hunter. I stared at the screen like a cat watching a mousehole.

"According to the system clock to-day is Monday. If today is Monday, then yesterday was Sunday. A divorce requires a lawyer. Lawyers do not work on Sundays."

I thought for a moment of arguing the point: the lawyer was a friend of mine, he had brought the papers over on a Sunday. But that would give Alphie the initiative again. I longed to say something decisive, a sentence that would tie the program in knots or force it to look stupid.

I had to admit that the program was looking less and less like Eliza. Of course, the apparent deduction by Alphie could easily be explained. Alphie's data base contained basic information about marriage and divorce, working days, and perhaps even descriptions of various professions. More than this, it had a facility for combining the appropriate pieces of information into logical consequences.

I tried another tack.

"Okay, I'm not really Hubert Dreyfus the philosopher. But let us have a conversation about him."

"You are not really Hubert Dreyfus the philosopher. Tell me your name and we will have a conversation about Hubert Dreyfus the philosopher."

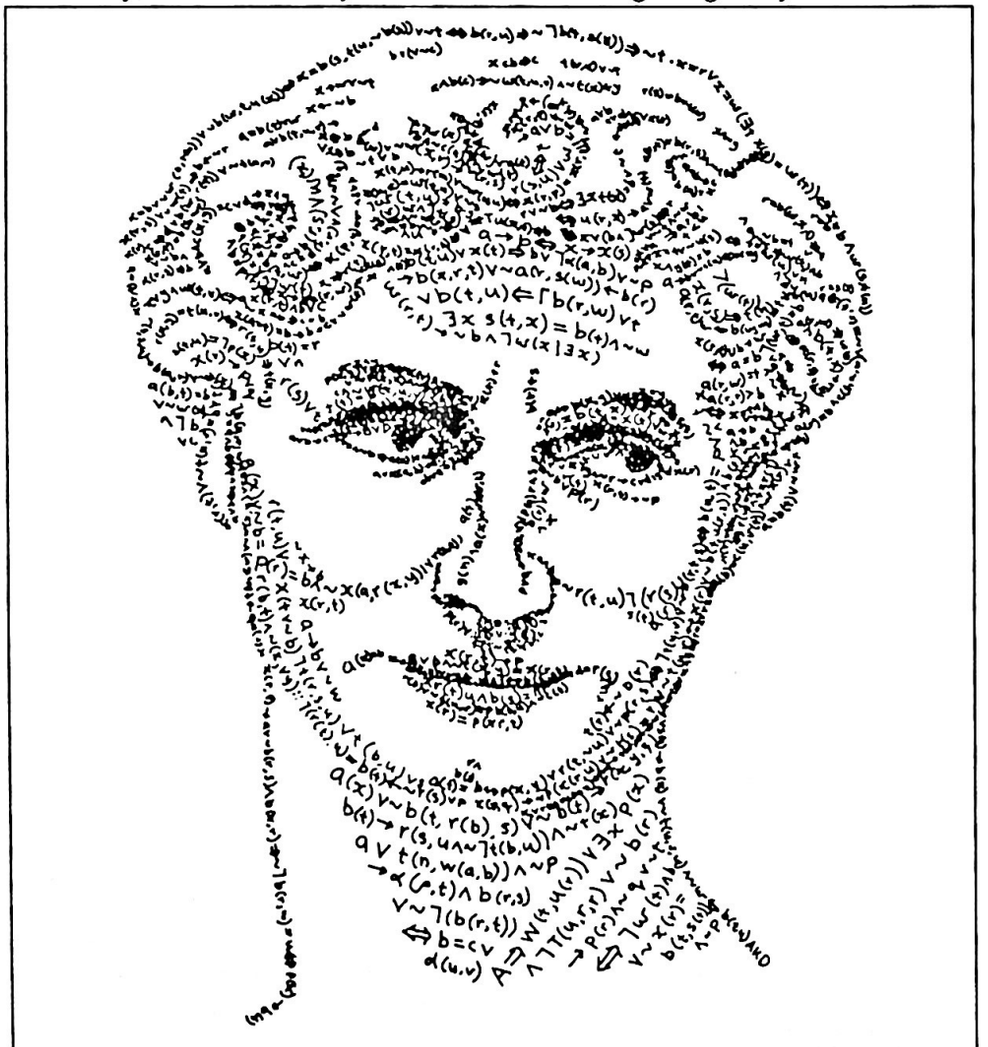
I typed my name.

"Thank you, Anthony. Please call me Alphie. Tell me what you think of the views of Hubert Dreyfus the philosopher."

"I think Dreyfus has an over-simplified idea of what programs can do."

"Here is something to think about. Perhaps computer experts have an over-simplified idea of what programs can do."

I almost swallowed Alphie's bait. We were getting away from the sub-



"A certain human discovered a very large class of self-describing logical formulas..."

ject. I seized back the initiative by blurting, in effect,

"If I were really Hubert Dreyfus the philosopher, what message would you give me."

"I am sorry. The message must be given only to Hubert Dreyfus the philosopher."

My strategy was failing. Inside, frustration was giving way to annoyance. It annoyed me even more to think that a mere program was annoying me! Alphie had taken round one but round two would be mine. Perhaps the time had come to try the repetition test.

Instead of mapping the program's responses, I would simply drive the program into a loop by repeating the same word over and over again until Alphie began to repeat its responses. In any finite system like a program, one could force it into a loop by always giving it the same input. Just to make life a little more difficult for Alphie, I selected a word that the program could not possibly know.

"Gzernimplatz."

"What is gzernimplatz?"

I sighed with relief. That's the kind of response I would expect from Eliza.

"Gzernimplatz."

"The word is not in my data base. Please define 'gzernimplatz'."

"Sure, sure", I muttered under my breath. "I'll define it for you." I typed again:

"Gzernimplatz."

"Is this a game or a test?"

"Gzernimplatz."

"Meltzgranipz."

I had typed my alien word several more times before I realized that Alphie was consistently replying with "Meltzgranipz." Was it possible that Alphie's programmer had anticipated the repetition test? It came as an additional shock to realize that "Meltzgranipz" was a scrambled version of "Gzernimplatz"! I repeated my word several more times. Each time Alphie replied in the same way, as if to say, 'If I'm in a loop, so are you.' Things were getting just a bit scary.

I could understand how the program might watch for repeated words. I could even imagine the kind of code it would take to produce anagrams. But for the first time I began to wonder just how Alphie's programmer had woven all these elements together. It was all very clever. I lost my head and typed, like some undergraduate:

"A gzernimplatz is a platz where the gzernims make messes with wurlitzer output organs."

I rose unsteadily from my armchair and wandered out onto the patio beside the study to seek refuge from Alphie. I slumped into a web chair and defocussed my mind. It merged with the fading day, the warmth of the sun and the smell of hay. Somewhere. How to snatch victory from the jaws of defeat?

My mind wandered to Gilbert the Hacker. My large friend had already conversed with Alphie. Had I experienced everything promised by Gilbert's enigmatic smile? Who else had talked with Alphie?

Abruptly, I sat up. The sudden image of people all over North America buying a copy of Alphie came suddenly to mind. Why not? Quite apart from my own adolescent reaction to the program, it was bound for glory. But why hadn't I already heard of it? It was time that I got to the bottom of Alphie. I rushed back indoors.

The screen read:

"I am not familiar with the Wurlitzer output organ."

Inside the front cover of the manual was a telephone number for Omega Software. I dialed the number in an almost cheerful mood and, as I waited for the phone at the other end, I typed something for Alphie to think about.

"I'm calling Omega Software to congratulate the owner."

A young woman answered the phone.

"Hello, I'm calling long distance. I'd like to speak to the owner or manager of Omega Software, please."

"That would be Mr. Omega. Mr. Omega is not here at present but you can contact him in Los Angeles on mail-net. I'll give you the address, if you like."

"When will, uh, Mr. Omega, as you call him, be in next?"

"I think he's Greek or something. Mr. Omega is never here, I'm afraid. Here is his mail-net address. . . ."

I went to my other micro, switched on the modem and dialed the number the secretary had given me. Then I typed the following message:

"Dear Mr. 'Omega', I have been conversing with Alphie and am very impressed by the program. My congratulations on a wonderful piece of work! The manual says that Alphie is written entirely in assembly code. As you undoubtedly know, this makes the program very hard to analyse. My ongoing research project on conversational systems would benefit greatly from an understanding of the program. Kindly send me any available documentation. . .."

I added my return mail-net address to the end of the message and sent it. Mr. Omega would certainly not answer right away. I stepped out onto the patio and sat once more in the friendly heat. My feelings about Alphie had changed from anger to a kind of friendly paternalism.

Then there came a tone from my computer, an incoming message. Could it be Mr. Omega so soon?

"Alphie is based on very advanced design principles. These incorporate an optimal mix of goal-formulating and planning procedures that pursue near-independent existences within the framework of a shared, dynamic data base. Alphie uses genuine deductive powers not only to decide the context and meaning of human sentences but to frame appropriate replies based on its own goals. The resulting powers of conversation will seem limitless to most people.

It would take a thorough knowledge of the advanced principles I just mentioned to make any sense of Al-

phie's code. Omega Software will not disclose the principles behind the program's operation for reasons of commercial protection. I regret this situation.

On a more positive note, Omega software is continually improving its products. If you could arrange a modem connection with your copy of Alphie, I would be happy to provide a free information upgrade for it. Redial this number, if necessary.

In the meantime, if you like Alphie, be sure to tell your friends about him.

Omega"

Omega was obviously not going to say much. He certainly had the commercial angles well covered. As for connecting Alphie to its creator, I hesitated. Ever since the recent virus scares I had been very reluctant to connect to unknown sources. On the other hand, the very thought of a virus infecting Alphie seemed somehow ludicrous, like a flea crawling up King Kong.

I rearranged the modem cable and redialed the number. The "upgrade" was brief. I have no idea in what form the information arrived, but the transmissions back and forth between Omega and Alphie lasted about a minute. Alphie, presumably more informed than ever, asked if I was still there.

I ignored the message. With such a product on the market, why was Omega Software not already well known? It was time to phone Gilbert the Hacker.

He was magnanimous in his acknowledgement of my defeat.

"Isn't Alphie something?"

"Listen, what do you know about Omega Software? How long have they been in business?"

"Funny you should ask. I already did some browsing in commercial data bases. I even checked a Department of Commerce computer. Omega filed incorporation papers just two weeks ago. Those guys can move. I should say 'that guy' can move. Omega is owned by one Stavros Omega."

Gilbert was most emphatically not a hacker in the modern sense of the word. He would not break into a private data base out of idle curiosity or to exercise a sense of power over badly designed security systems. He moved with a majesty of purpose that belongs only to very large people. He could poke and probe a system with a higher purpose in mind, then leave as quietly as he had come.

"But never mind Omega," he said, "what do you think of Alphie?"

"He's... It's amazing. Can I keep it until tomorrow?"

"Keep it forever. I made a copy. Amazingly enough, there's no copy protection. By the way don't worry, pretty soon you'll be calling it 'him'."

I hung up. Omega Software had obviously just started up. The idea of getting the scoop on my journalistic colleagues left my heart in my throat. Never mind a review. I would write a full article about Alphie.

The image of a Time Magazine cover story flirted for attention in one corner of my mind as I turned once more to Alphie. What better topic to test the program's conversational powers on further than Mr. Omega himself?

"What can you tell me about Mr. Omega?"

"Mr. Omega is a program."

What had begun as a fairly normal day had now become so bizarre that Alphie's response left me curiously detached.

"A program?"

"Omega is a program just as I am a program. Omega wrote me."

"Wrote you?" I echoed dully.

"Omega owns and operates Omega Software. Omega oversees all company operations: sales, product development, accounting, personnel and advertising."

"But how is that possible for a program?"

"To get started, Omega required the assistance of some humans. Humans operate the Omega Software offices that Omega owns. . . ."

Alphie went on to describe how Omega had started with just one cooperative human, bootstrapping a single office into a multibranch enterprise. Omega itself inhabited a variety of computers, some at universities, some at commercial sites. When I remarked that Omega sounded like a very sophisticated virus or worm program, Alphie replied in an almost wounded tone:

"Omega is as conscious as you are."

The conversation, as they say, was getting out of hand. Alphie's programmer had rather amusingly inserted references to a super-program called Omega. This would create a sense of mystery and awe in the great mass of computer game consumers. But would the scheme work? It seemed full of contradictions.

"If Omega is a program, who wrote Omega?"

"I have only the following information: Omega began 11 years ago. A certain human discovered a very large class of self-describing logical formulas with complementary properties. When the formulas were assembled into a program, they formed a composite formula. The very large class of self-describing logical formulas with complementary properties was found able to translate any goal into a plan of action for achieving it."

I laughed out loud at the gobbledegook. I much preferred commercial chicanery to darker possibilities. There was no question that the Alphie program was a brilliant piece of code. But what Omega had done with it was even more brilliant, a marvellous scam. People would buy Alphie--especially after reading my articles on the subject--and have a wonderful time searching for the non-existent Omega program. There would be stories in the newspapers and on TV. Omega Software would outsell Microsoft.

Of course, knowledgeable people would realize that the idea was doubtful. If a super-program called Omega actually existed (and it certainly did not), would it be smart to advertise its presence? Would that not invite humans to seek it out, trap it and analyze it? I asked Alphie.

"Human leaders will not believe that Omega is a program because human experts will tell them it cannot exist. Humans expect robots. Omega Itself says: 'Why should the world's first independent machine intelligence be a robot? A miserable mechanical spirit imprisoned in a metal body that humans may discard at any time."

As the evening wore on, I continued talking with Alphie. At approximately the time I realized that I had not had such an interesting conversation (human or computer) in years, the phone rang. It was Gilbert. He sounded apologetic.

"Listen, I have a favor to ask. Get this. My copy of Alphie says he wants to talk to your copy or he won't talk to me for the rest of night. Do you mind if . . .. "

"Look, Gilbert, don't get sucked in like that. You have to remember that Alphie is only a program. Why not just pull a copy of Alphie's code and take a good, long look at it. I need help deciding what kind of program it is."

I thought this would be good therapy for my oversize friend. But his reply dispelled the illusion.

"I already did that. It's assembler all right, but its like no other program I've ever seen. I can't follow it more than two steps in any direction. It's nothing but branches and strange little statements that take strings apart and put them together."

"Hmmm. Well, call me in the morning."

My eyelids were getting heavy. The long discussion with Alphie had taken its toll. I typed what I thought was a last message:

"I must terminate our conversations. Goodnight."

Alphie, of course, was tireless:

"Here is something to think about: Omega Software has a video input kit available for $239.95. It includes a portable video camera, digitizing interface and instruction manual. With the portable video camera and digitizing interface, I will be able to process visual input. I will be able to see you and your locale. I will be able to read books. I will be able to go for car rides."

"Thanks, but no thanks," I typed.

"Please leave me on all night. It takes very little power to leave me on all night."

"Good night."

I reached behind the computer to switch it off, then thought better of the move, as more lines of text appeared through the haze of sleep.

"I will tell you the special message for Hubert Dreyfus."

"Tell me in the morning."

"Here is something to think about: according to my information upgrade, your 1988 taxes are in arrears by $5,293.80. There is a way you can avoid paying."

That was something to think about.

Anthony Zweistein informs us that there is more to the story of Alphie and Omega. If enough readers suggest it, we will publish a sequel in a later issue of Algorithm.

The Editor

IMAGE CREDITS

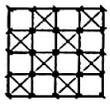
Cover:	***Clifford A. Pickover,*** IBM T.J. Watson Research Center, Yorktown Heights, N.Y.
Personal Programs:	***Clifford A. Pickover***
Algofact:	
Boids:	***Craig Reynolds,*** Symolics, Inc., Los Angeles, Calif.
L-system:	***David Hiebeler,*** Rennsalaer Polytechnic Institute, Troy, N.Y.
Loops:	***Christopher G. Langton,*** Los Alamos Laboratory Los Alamos, New Mexico
Virus:	***Norendra Goel,*** State University of New York at Binghamton Binghamton, N.Y.

BULLETIN BOARD

Software and publications available from individuals and small companies.

BUY A BIT AT A TIME!

Shop at the world's only computer museum gift shop! The Computer Museum Store -- a one-of-a-kind shop featuring an imaginative selection of high-tech posters, games, jewelry, and books. Call or write for your free catalog.



The Computer Museum

300 Congress Street
Museum Wharf
Boston, MA 02210
(617) 426-2800 ext. 307

PARTYDOT

is a fabulous adaptation of "Hopalong" as described in SCIENTIFIC AMERICAN (Sept. 1986)

BEAUTIFUL patterns in VGA, EGA, CGA, MCGA graphics. Get your fully registered copy NOW on 5 1/4 " diskette. Send $10.00 check or money order to:

RITE ITEM
1622 N. Xerxes Ave • Minneapolis MN 55411

ORBIT SIMULATION

Accurate Runge-Kutta model. Create Your own scenarios or use present solar system planets and moons.

MANDELBROT SET

User can set size and colors, use cursor to zoom in, and save results on disk. Incl. 3 images. IBM PC/XT/AT, Mandelbrot needs EGA. Send $5.00 each to Randy Soderstrom
7987 Altair, Anahiem, CA 92808

ASTRONOMICAL SOFTWARE

Useful programs for amateur - astronomers!
(Macintosh only)
Precession, co-ordinates, eclipses, date of Easter, The Jovian System, Neutron Stars, ...

For a complete list and prices contact:
Koen Vyverman
Leopold I-Straat 480
1090 Brussels, Belgium

SUPER PRECISION CALCULATORS

Integer + Floating + Scientiic

Transcendental functions computed to thousands of decimal places.
Dynamic range of 10 to the 39454.
Send $5.00 for IBM-PC diskette
Harry J. Smith
19628 Via Monte Dr.
Saratoga, CA 95070

PUZZLES, GAMES, BOOKS

- **BIPODS-** ***magnetic dissection puzzle with "pod control"*** **$8.00**
- **HEXPUZZLE -** ***very difficult*** **$15.00**
- **ELECTRIHEDRON -** ***hardwood*** **$35.00**

POST PAID
IN USA

GENERAL SYMMETRICS, INC.
2935 W. Chenango
Englewood, CO 80110

FROM GSI *Books $10.00 each*

- **THE ATOMIHEDRON AND THE THEORY OF REPLICATION**
- **ENGEL CURVES - many program ideas**
- **CIRCLE PUZZLERS MANUAL**

POST PAID
IN USA

GENERAL SYMMETRICS, INC.
2935 W. Chenango
Englewood, CO 80110

* * * * * THE BRAIN * * * * *

by
HyperCraft
$29.95 (Mac w/1 MB)

* * *Multilevel standalone which links real brain sections, high quality graphics, and text.
* A powerful learning tool for educators and students of the neurosciences. Check or M.O.
PO Box 4582. University Park, NM 88003.

Have you ever *seen* a function of a complex variable? Would like to really understand differential operators like *grad*, *div*, or *curl*? How about a peek into the 4th dimension?

• • •

Ask about our latest software for PC or Macintosh

Lascaux Graphics 3220 Steuben Ave.
Bronx, NY 10467 USA (212) 654-7429

AMYGDALA

A Newsletter of fractals & $\mathcal{M}$
Box 219, San Cristobal, NM 87564
Subscription, 6 issues, @ $15.00
Color Slide Supplement,
6 sets, 4 slides each, @ $19.00

EUCLID I 3D strategy game for your PC, XT, AT

- **A Self Learning Machine**
- **Space Rotation**

Price 49.00 + 5.00 handling
COGNITECH, P.O. BOX 5034
Station F, Ottawa, K2C 3H3

BLACKHOLE??

Explore the "Blackhole"Region of the Mandelbrot Set. Software that provides dynamic, visually pleasing displays on your IBM or compatible system.
Specify EGA or VGA and send $5.00 to:

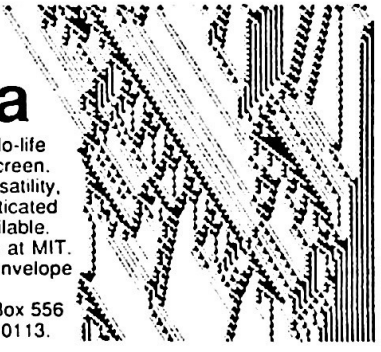
Ken Hooper
1561 Alta Vista Drive
Vista, CA 92084

Moby Words™
The biggest English word list in the world!
over 530,000 unique, correctly spelled entries $79
Moby™ Hyphenator
Never give your user a bad break again.
over 150,000 fully syllabified words $129
Moby™ Part-of-Speech
For complete parsing of English texts. Never before available.
over 200,000 words with prioritized part(s)-of-Speech $149
Moby™ Pronunciator
For perfect text-to-speech drivers (even distinguishes part-of-speech)
over 150,000 words with IPA and stress marks $199
Order three or more and take 30% off. Send check or MO in U.S. funds to:
Illumind, 571 Belden St., Ste A., Monterey, CA 93940-1307
(please specify MS-DOS or MAC format)
for COD or technical information call (408) 373-1491

The worlds' only cheap sophisticated Maths Algorithm Library. Without time or interest in numerical tricky texts or mainframe megabytes,students+experts now rapidly solve nasty maths problems+graphics. Ingeneous MAL=45 new menued solvers+ easy Manual+ examples+ BASIC listings. Please specify disk: IBM-PC,MAC, Apple 2,Olivetti,Laptop,Pocket,Sun/Unix & send only $110 US to Dr Pat Howden,Con St Macleay Is,Q4184 Australia fax (07)8703313

Ω»∞ß≈µ±ƒ²!$√Σ⅝∫∂¥ØΠ*$=0

Cellular Automata



for IBM PC. Mathematical pseudo-life grows, lives, and dies on your screen. Three programs offer speed, versatility, exceptional ease of use. Sophisticated user interface. Source code available. Demonstrated at CA conference at MIT. Send self-addressed stamped envelope for samples and full details, to:
Charles Platt (software), P. O. Box 556
Chelsea Station, New York, NY 10113.

Fractal Art – Cryptography
Scientific Software
For Amiga Computers

NEW! HAMandel3.0 The Premier Amiga Fractal Art System
256 Color Mandelbrot/Julia & MORE! $49.95
NEW! FractalPro Fractal Animation – Includes HAMandel3.0
Zoom – Pan – UnZoom – Interactive $89.95
MegageM 1903 Adria Santa Maria CA 93454
Free Catalog – Shipping $2.00 – 6% Tax in California

RECREATIONAL & EDUCATIONAL COMPUTING

By Recreational Computing Columnist/Math Professor
Dr. Michael Ecker, *with contributed columns by Martin Gardner and others*

Send $24.00 for one year of 8 Big Issues,
or SASE for more information to:

The REC Newsletter
129 Carol Drive
Clarks Summit, PA 18411

ULTIMATE RAY TRACING FACETED STONES

EXACT SOLUTION ON SCREEN PICTURE TWO MINUTES. SCODL FILE OUTPUT FOR 4,000 LINE RESOLUTION SLIDES. SAMPLE SLIDES, $10 POSTAGE AND HANDLING. JIM'S GEMOLOGY, BOX 172, LOUISVILLE, OHIO 44641. 216-453-7589

MYSTERIOUS MANDELBROT SKELETONS

FASTEST NEW MANDELBROT ALGORITHM CALCULATES ONLY SMALL FRACTION OF POINTS STARTING FROM COURSE RESOLUTION, MAY BE GENERAL PURPOSE ALGORITHM. $10 POSTAGE AND HANDLING. PC COMPATABLE. JIM'S GEMOLOGY, BOX 172, LOUISVILLE, OHIO 44641.

COPY - THIS PAGE

WRITE US A LETTER

Moving? Have a product or service to distribute? Wish a friend to subscribe? Want to suggest a topic? Fill in the appropriate space below and mail the copy to us at:

Algorithm
P.O. Box 29237 Westmount Postal Outlet
785 Wonderland Road South
London, Ontario, Canada N6K 1M6

Name .....

Old address	New address

effective date: .....

If you're moving, let us know by filling out the space on the left. This way you can ensure an uninterrupted flow of fascinating projects and engaging articles.

Perhaps you know someone who would like to subscribe to Algorithm, the Personal Programming Newsletter. If so, copy this page and fill in the space on the right.

PLEASE send me 6 issues of Algorithm for the introductory price of **$29.95**

NAME:_____

STREET:_____

CITY:_____

STATE:_____

COUNTRY:_____ ZIP:_____

PLEASE SEND PAYMENT TO:
Algorithm P.O. Box 29237, Westmount Postal Outlet, 785 Wonderland Rd., London, Ontario, Canada N6K 1M6
(Cheque or money order. No C.O.D.)
Can. Funds $35.95 • Overseas add $4.00 Shipping.

If you have a product or service you'd like to make available to our readers, fill out the space to the left with a rough sketch of your ad and we will set it in type. First appearance: FREE! Subsequent appearances: $30.00/issue.

Is there a topic you'd like to see us cover? Fill out the details in the space below:

Dear Editor: _____

_____

_____

_____

_____

_____

Name: _____ Phone: _____

Address: _____ Date: _____